Achieving Image Super-Resolution using CNNs and GANs

Fábio Faria Dias[1], Bruno Binkowski[1]

Keywords: Single Image Super-Resolution, SRCNN, GAN, Very Deep Convolutional Networks

1. Introduction

Single-image super-resolution (SISR) techniques are often the subject of research in computer vision. The main objective of such techniques is to recover a high-resolution (HR) image from a low-resolution (LR) one. SISR is an important subject of study due to its multiple applications, ranging from recovering images from surveillance cameras to applications in medical imaging. where having more details is often required on demand, as we can see in Kim et al. (2016).

The goal of our work is to investigate how to achieve SR using Deep Convolutional Networks (SRCNN) and Generative Adversarial Networks (SRGAN). We first start defining the problem we want to solve. Afterwards, we suggest different Deep Convolutional Neural Network model architectures to tackle the issue, pointing out the decisions we made during our development. In the end, we evaluate the results using Accuracy and Peak Signal to Noise Ratio (PSNR).

2. Image Super Resolution

We call super-resolution the task of estimating a high-resolution (HR) image from a low-resolution (LR) counterpart Ledig et al. (2017).

Super-resolution problems can be approached using a vast variety of methods ranging from interpolation and statistical methods to deep convolutional networks. Over the years, with the increase in performance and accuracy of modern CNNs, many studies have focused on proposing methods of estimating HR images using deep convolutional neural networks. These methods normally propose the use of CNNs to estimate a map between LR and HR images without recurring to feature engineering. Unfortunately, these approaches normally fail to add high-frequency details in the generated images, making their results normally blurry and visually unpleasant.

Some modern approaches include the use of Generative Adversarial Networks as an alternative to include texture details in super-resolved images.

3. Convolutional Networks for Super Resolution

Our work proposal uses as a reference three different approaches for Super-Resolution problems. These methods were posed and investigated in the studies mentioned in the following sessions.

3.1. Super-Resolution Convolutional Neural Network (SRCNN)

The first proposal is presented by Dong et al. (2016), that discusses the classical problem of single-image super-resolution in computer vision and introduces a groundbreaking solution called the Super-Resolution Convolutional Neural Network (SRCNN). Traditionally, example-based strategies were employed, such as sparse-coding methods, which involved complex pipelines. However, SRCNN offers a paradigm shift by directly learning an end-to-end mapping between low- and high-resolution images through convolutional neural networks.

Key features of SRCNN include intentional structural simplicity, superior accuracy compared to traditional methods, and impressive efficiency, even on CPUs. The entire super-resolution pipeline is learned, minimizing the need for extensive pre/post-processing. Dong et al. (2016) emphasizes three significant contributions: the introduction of a fully convolutional neural network for image super-resolution, the establishment of a relationship with traditional sparse-coding methods, and the demonstration of the efficacy of deep learning in solving the classical problem of super-resolution.

Building upon a preliminary version, the article enhances SRCNN with larger filter sizes and deeper structures, extending it to process three color channels simultaneously. Experimental results demonstrate improved performance, and the model outperforms existing methods across various evaluation metrics. SRCNN emerges as a transformative approach, offering simplicity, accuracy, and speed in solving the single-image super-resolution problem.

3.2. Accurate Image Super-Resolution Using Very Deep Convolutional Networks

The Kim et al. (2016) proposal addresses the challenge of single-image super-resolution (SISR), aiming to generate a high-resolution (HR) image from a low-resolution (LR) one. The evolution of SISR methods is discussed, from early techniques like bicubic interpolation to more recent approaches utilizing statistical priors or internal patch recurrence.

The focus then shifts to the current trend of employing learning methods, particularly Convolutional Neural Networks (CNNs), for LR-to-HR mapping. The Super-Resolution Convolutional Neural Network (SRCNN) is mentioned as an effective method that achieved state-of-the-art performance without the need for engineered features.

However, SRCNN has limitations, including reliance on small image regions, slow training convergence, and operation at a single scale. The proposed solution introduces improvements, such as leveraging contextual information over large image regions, speeding up training through residual-learning CNNs and high learning rates, and addressing multi-scale super-resolution within a single network.

In summary, the work presents a highly accurate super-resolution method based on a very deep convolutional network, overcoming SRCNN's limitations.

3.3. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGAN)

The Ledig et al. (2017) proposal discusses the complex task of enhancing a high-resolution (HR) image from a low-resolution (LR) version is known as super-resolution (SR). This area has garnered significant attention within the computer vision research community due to its broad applications. The inherent challenge in SR, particularly for high upscaling factors, often results in the absence of fine texture details in the reconstructed SR images.

Commonly, supervised SR algorithms aim to minimize the mean squared error (MSE) between the recovered HR image and the ground truth, optimizing for peak signal-to-noise ratio (PSNR). However, the limitations of MSE and PSNR in capturing perceptually relevant differences, especially in texture detail, are evident, as they rely on pixel-wise image differences.

To address these limitations, the proposed approach introduces a super-resolution generative adversarial network (SRGAN). This SRGAN utilizes a deep residual network (ResNet) with skip-connections and moves beyond MSE as the sole optimization target. Unlike previous methods, a novel perceptual loss is defined, incorporating high-level feature maps from the VGG network and a discriminator that encourages solutions perceptually challenging to distinguish from HR reference images.

The focus on perceptual loss aims to improve the photo-realism of the super-resolved images. An example of a photo-realistic image with a 4× upscaling factor is presented, showcasing the effectiveness of the SRGAN approach.

4. Proposed Work

We started our project using these works as references to implement our models. Before mentioning each model implementation, let's start defining the dataset we used during our development.

4.1. Dataset

We performed our experiments on a famous dataset called DIVerse 2K (DIV2K), being Agustsson and Timofte (2017) one of the articles that used the dataset. DIV2K has high-quality 2K images along with their low-resolution counterparts downgraded using scaling factors of 2, 3, 4 and 8. This dataset is widely used on super-resolution problems since it already has the LR and HR pairs prepared for use.

Figure 1: DIV2K dataset

4.1.1. Super Resolution CNN

Our SRCNN model was defined using the function in the next figure. The idea behind this method is to use a shallow CNN to approach SISR as a regression problem. As you can see, this network was defined using 2 as an upscale factor. Also, this model was compiled using mean squared error to calculate the loss, and the optimizer Adam to update the weights. Note that the model does not use any downsampling layer. This is because generative SR models should add information to the image while preserving as much information as possible about the original LR image.

```
def srcnn_model(input_shape, upscaling=(2,2)):
    input_layer = Input(shape=input_shape)

    # Feature extraction
    x = Conv2D(64, (9, 9), activation='relu', padding='same')(input_layer)

    # Non-linear mapping
    x = Conv2D(32, (1, 1), activation='relu', padding='same')(x)

    # Upsampling
    x = UpSampling2D(size=upscaling)(x)

    # Reconstruction
    x = Conv2D(3, (5, 5), padding='same')(x) # No activation on the last layer

    model = Model(input_layer, x)
    model.compile(optimizer='adam', loss='mean_squared_error', metrics=["accuracy"])

    return model

srcnn = srcnn_model(input_shape=(128,128, 3))
```

Figure 2: SRCNN

We trained this model using batches of size 32 and 80 epochs. For our tests, we defined this model with an input shape of (128, 128, 3). The idea was to use small images to have quicker results.

4.1.2. Very Deep Convolutional Networks (VDSR)

This method was proposed by Kim, Jiwon et al.. It consists of increasing considerably the number of convolutional layers when compared to SRCNN models. The idea behind this is to give the model more trainable parameters to fit better the manifold during the training.

See in the next figure how the model was created. Observe how this model can have a bigger number of convolutional layers stacked together. Also, observe how the image can be upscaled on the transposed convolution function.

We also compiled this model using mean squared error to calculate the loss, and the optimizer Adam to update the weights.

```
def vdsr_model(scale_factor=2, num_filters=64, num_layers=20):
    input_low_resolution = tf.keras.Input(shape=(None, None, 3))
    # Initial convolution
    x = layers.Conv2D(num_filters, 3, padding='same', activation='relu')(input_low_resolution)
    # Intermediate convolutions
    for _ in range(num_layers - 2):
        x = layers.Conv2D(num_filters, 3, padding='same', activation='relu')(x)
    # Upsampling layer
    x = layers.Conv2DTranspose(num_filters, 3, strides=scale_factor, padding='same', activation='relu')(x)
    # Final convolution
    output_sr = layers.Conv2D(3, 3, padding='same')(x)  # No activation for the last layer

    model = Model(inputs=input_low_resolution, outputs=output_sr)

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=["accuracy"])

    return model

vdsr_model = vdsr_model(scale_factor=2, num_filters=64, num_layers=20)
vdsr_model.summary()
```

Figure 3: Very deep super resolution

4.1.3. Generative Adversarial Networks (GANs)

The methods proposed by Ledig et al. (2017) aim to approach the SR problem both as an adversarial and as a regression problem. The idea is to have a model able to generate SR images (generator) and a discriminator model able to distinguish between real or fake (generated) images. Combining both of them during the training can help us to make the generator better at generating realistic images, as well as to make the discriminator better at his binary classification task. Our GAN model uses a VGG19 network pre-trained on Imagenet to perform feature extraction and help us with the regression problem. The difference here is that we are not performing a pixel-wise regression. Instead, we will use the learned features from the 10th convolutional layer in VGG19 to help with the generator training. The next figure 4 shows a brief overview of how the models were defined.

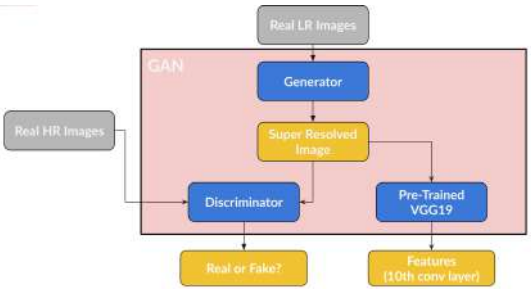

Figure 4: GAN Block Diagram

See that the discriminator is used to classify if an image is super-resolved or has a high resolution. The generator creates the super-resolved images and feeds both the discriminator and the pre-trained VGG19 with them.

Figure 5 shows a brief overview of how the model is trained. The idea is to perform it in two steps. The first one is to feed the discriminator using HR and SR images, using them to calculate the loss using binary cross entropy. This loss is then used along with an optimizer (adam) to calculate the discriminator weights. Then we proceed with the generator training loop, which is done indirectly using the GAN model. This model was defined to indirectly train the generator model using the outputs of the discriminator and the pre-trained VGG19. Observe that both discriminator and VGG19 are kept with non-trainable parameters during this stage. We use LR and HR images to feed the GAN, which uses adversarial and content loss to update the generator weights. The content loss is calculated using mean squared error and the weights are also updated using Adam's optimizer.

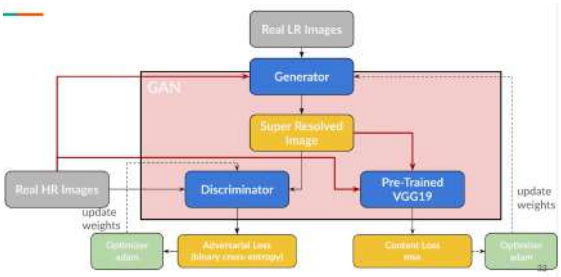

Figure 5: GAN - Training Loop

See in Figure 6 how the generator was defined. The residual blocks have two Convolutional layers using ReLU and PReLU as activation functions, respectively. These blocks also use batch normalization.

```
def generator_model():
    input =  tf.keras.Input(shape=(None, None, 3))

    # Encoder
    x = layers.Conv2D(64, (9, 9), padding='same')(input)
    x = PReLU(shared_axes = [1,2])(x)
    temp = x

    for _ in range(32):
        x = residual_block(x)

    # Decoder
    x = layers.Conv2D(64, (3, 3), padding='same',
activation=tf.keras.layers.LeakyReLU(alpha=0.4))(x)
    x = layers.add([x, temp])
    x = upsampling_block(x)
    output = layers.Conv2D(3, (3, 3), padding='same',
activation=tf.keras.layers.LeakyReLU(alpha=0.3))(x)

    model = Model(inputs=input, outputs=output)

    return model
```

Figure 6: Generator Function

Observe in Figure 7 how the discriminator was defined. It consists of multiple Convolutional blocks using k=(3x3) and leaky ReLU as an activation function. See that this model ends with a binary classifier.

Last, Figure 8 depicts how we instantiate our GAN model. Observe that this model receives two different inputs (LR and HR imageS) and outputs a classification and the VGG19 features.

```
# Discriminator model
def discriminator_model(input_shape):
    input = tf.keras.Input(shape=input_shape)

    features = 32

    x = discriminator_block(input, features, batch_norm=False)
    x = discriminator_block(x, features, strides = 2)
    x = discriminator_block(x, features * 2)
    x = discriminator_block(x, features * 2)
    x = discriminator_block(x, features * 4)
    x = discriminator_block(x, features * 8)
    x = discriminator_block(x, features * 8)

    x = layers.Flatten()(x)
    x = layers.Dense(features)(x)
    x = LeakyReLU(alpha=0.2)(x)

    output = layers.Dense(1, activation = 'sigmoid')(x) #Output a single valu
for real or fake

    model = Model(inputs=input, outputs=output)

    return model
```

Figure 7: Discriminator

```
def gan_model(train_inputs, train_targets, generator, discriminator, vgg):
    low_resolution_input = tf.keras.Input(shape= np.shape(train_inputs[0]))
    high_resolution_input = tf.keras.Input(shape=np.shape(train_targets[0]))

    generator_output = generator(low_resolution_input)
    generated_features = vgg(generator_output)

    discriminator.trainable = False  # Freeze discriminator during GAN training

    discriminator_output = discriminator(generator_output)
    model = Model(inputs=[low_resolution_input, high_resolution_input],
outputs=[discriminator_output, generated_features])

    return model
```

Figure 8: Discriminator

5. Experimental Results

5.1. SRCNN

See in Figure 9 and Figure 10 the results for the SRCNN accuracy and loss after 80 epochs. Observe that the accuracy for the two datasets (training and validation) reaches a peak around the epoch 70.

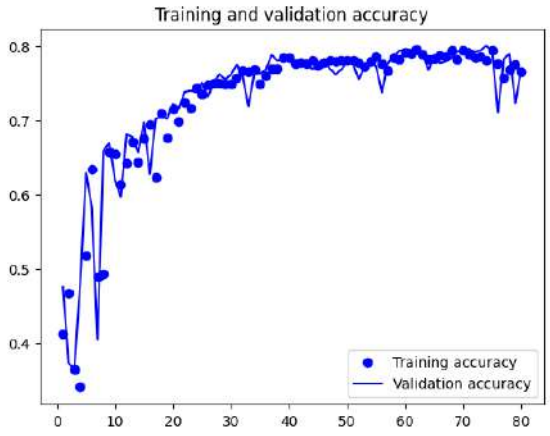

Figure 9: Accuracy during training (80 epochs). SRCNN Upsampling - Input Size 128 x 128 x 3. Output size 256 x 256 x 3.

Using the trained model to generate SR images from an unseen LR image, we obtain the images in Figure 11. Our results show a satisfactory result in generating an upscaled image but with some blurred aspects added by the model. It leaves us to conclude that our model is acting as a high-frequency filter.

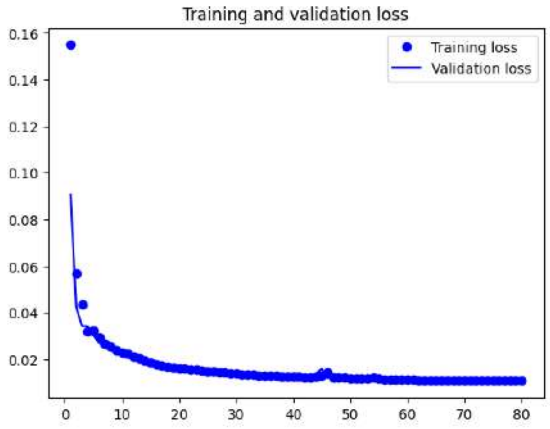

Figure 10: Loss during training (80 epochs). SRCNN Upsampling - Input Size 128 x 128 x 3. Output size 256 x 256 x 3.

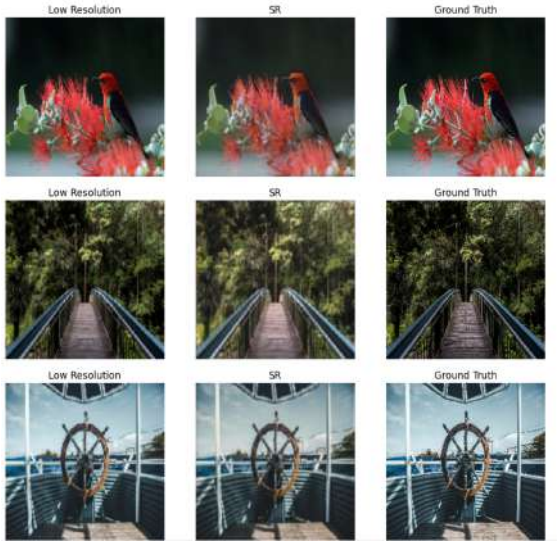

Figure 11: Inference on 3 sample images after 80 epochs of training. The first column shows the low-resolution input. The second column shows the resulting image after inference. The third column shows the ground truth.

5.2. Very Deep Super Resolution (VDSR)

We trained our model for 150 epochs optimizing the loss using MSE (Figure 12. See the two samples of generated images in figures 13 and 14. Observe that the model was able to regress a higher resolution image, but maintain the same blurry aspect of the SRCNN model.

It leaves us to conclude that besides these models can show good results on accuracy and loss, they don't produce realistic details since the model was never trained to achieve this goal.

5.3. Generative Adversarial Networks (GANs)

Figures 15 and 16 show SR images generated after 5000 epochs of training. Using this model has shown to be effective in the task of adding details and texture to the generated SR figures.

Our models have shown a potential to improve regarding the generator loss. We have faced a lot of difficulties in decreasing

Figure 12: Accuracy, loss and PSRN for after 150 epochs of training

Figure 13: Inference on 1 sample image after 100 epochs of training. The left figure shows the low-resolution input. The right figure shows the resulting image after inference.

Figure 14: Inference on 1 sample image after 100 epochs of training. The left figure shows the low-resolution input. The right figure shows the resulting image after inference.

these values for values smaller than 0.2e-06. Some adjusts in the network should be explored in order to fix this problem.

Figure 15: Inference on 1 sample LR image after 5000 epochs of training. The left figure shows the low-resolution input. The right figure shows the resulting image after inference.

Figure 16: Inference on 1 sample LR image after 5000 epochs of training. The left figure shows the low-resolution input. The right figure shows the resulting image after inference.

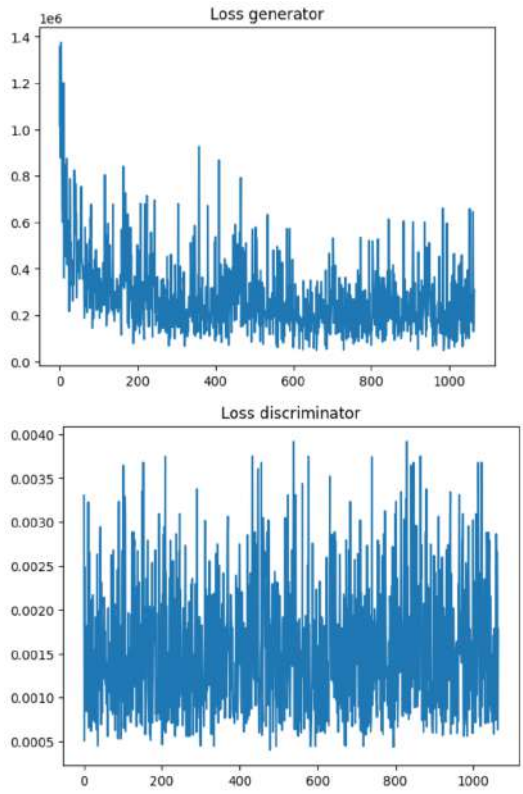

Figure 17: Adversarial and Generator loss after 1000 epochs

6. Conclusions

During our work, we observed that the choice of model architecture is very important in the final results of a SISR task. Even though the models SRCNN and VDSR have a large number of trainable parameters, they fail to create SR images with details and textures, giving a very poor quality when we take into account our visual perception. Metrics such as accuracy and loss might be optimized using these approaches, but in the end, the visual perception is not good.

In counterpart, GANs have shown very good results both on up-scale and texture creation tasks for SISR. This approach adds some complexity to the training procedure, which can be harmful during the development stage. In the end, our results

have shown good potential to be explored deeply.

7. Future Works

Things that can be done to improve our results:

- Use bigger image resolutions to train the networks;
- Use residual connections on VDSR networks;
- Adjust the Generator to decrease the content loss: Our model stops learning after 500 epochs. We could explore some techniques such as residual connections, changing activation functions, etc to allow it to have a smaller loss;
- Research and employ better metrics to evaluate the models.

References

Agustsson, E., Timofte, R., 2017. Ntire 2017 challenge on single image super-resolution: Dataset and study, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops.

Dong, C., Loy, C.C., He, K., Tang, X., 2016. Image super-resolution using deep convolutional networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 38, 295–307. doi:10.1109/TPAMI.2015.2439281.

Kim, J., Lee, J.K., Lee, K.M., 2016. Accurate image super-resolution using very deep convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., Shi, W., 2017. Photo-realistic single image super-resolution using a generative adversarial network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).