Generative Adversarial Networks for Image Upscaling

Ammar Rasyad Chaeroel (13521136)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): ammarasyad@gmail.com

***Abstract*—The balance between image quality and storage space/bandwidth is a long-standing issue to balance. To have higher image quality, more storage space is needed, therefore if the image is shared across the internet, more bandwidth is needed. Old images are also an issue, where image quality was not up to par with current era technology. Researches on image upscaling are currently ongoing, with technology and artificial intelligence maturing. The use of generative adversarial networks in image upscaling compared to convolutional neural networks are starting to pick up pace, with better results and efficiency.**

Keywords—image processing, artificial intelligence, generative adversarial networks, image upscaling, image super-resolution, image restoration

I. Introduction

An image is a way to convey information to the public. Information that cannot be conveyed through text or audio. It is important that this information can be received well and the point of the message is clear. One way of ensuring this is image quality, more importantly image resolution. Aside from images purely for comedic purposes shared on the internet, blurry or pixelated images can hinder the process of spreading information.

It is not as easy as capturing or creating images with the highest resolution possible. One of the most common vectors for sharing images, the internet, has wildly varying bandwidth. People may not have the fastest internet connection to download multiple large images. It is important to make sure that images are also accessible. There are cases where the source image simply does not have a high resolution, as is usually the case for images from decades ago. To solve cases where the source image does not have sufficient resolution, image upscaling is used.

Image upscaling is the act of increasing the resolution of an image by interpolating details from existing details on the image, instead of simply resizing the image to a higher resolution, which has the opposite effect of making the image blurrier. One method of upscaling is called bicubic interpolation, which is not always ideal. Bicubic interpolation has its limits, and at a certain point the amount of detail "created" is not significant to improving image quality. In recent times, better methods of upscaling have been discovered through the use of artificial intelligence, particularly using a Generative Adversarial Network (GAN).

Generative Adversarial Network is a machine learning model that can be used to generate new details. In such network, two neural networks contest with each other in a zero-sum or a min-max game, where one network's "win" is the other's "loss." The winning network is called a generator, and the losing network is called a discriminator.

A generator is the neural network in which its input and output are set up manually, while a discriminator is exactly what the term implies—it functions as a classifier. For image upscaling, the generator tries to generate an upscaled image, or rather generates the additional details to upscale an image, while the discriminator distinguishes upscaled images from real images.

With GANs, while it is not the only way to effectively and efficiently upscale an image, low-resolution images can be enhanced to make an image look better.

II. Theoretical Basis

A. Images and Super Resolution

Images have a specific resolution with its own file format. Each image file format has its own way of encoding information to minimize space while maintaining image quality, or at most compromising details that are visually indistinguishable. Formats like JPEG, PNG, WebP, HEIC/HEIF, AVIF, etc. have their own quirks, namely in their compression and encoding methods.

JPEG is the most commonly used image file format, both on the internet and what's commonly produced by cameras. A phone camera outputs a JPEG file, although most modern phones have the option to switch to HEIC for better compression and image quality relative to its storage space consumption. JPEG uses a lossy compression method, which sacrifices a tiny bit (almost visually indistinguishable) of detail, of which can never be recovered when decoded. PNG on the other hand uses lossless compression, where no detail is lost. The cost of lossy compression file formats is compression artifacts. In the case of lossless compression, it is storage space.

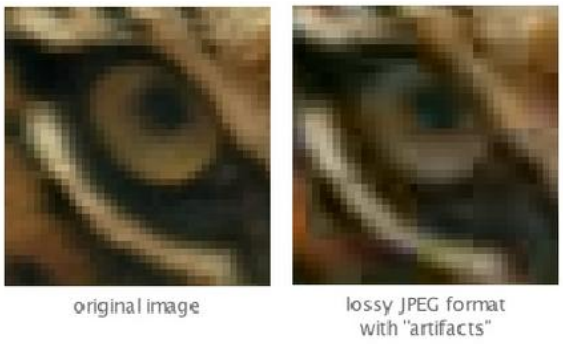

Figure 1 JPEG compression artifacts

Super-resolution or image upscaling is a way to reduce these artifacts and other image degradations to increase image quality—in other words, increase the resolution of the image. While increasing image quality does not always mean increasing image resolution, super-resolution aims to achieve both. The basic method is bicubic interpolation, where it interpolates new pixel values based on a weighted average of neighboring pixels. While it may achieve great image quality in certain cases, it doesn't apply to every case. There are limits to bicubic interpolation, and with artificial neural networks, another way is formulated to better increase image quality.

B. Generative Adversarial Network

As previously mentioned, a generative adversarial network is an artificial neural network model that can be used to generate new details, in this case for image upscaling. It is not the only neural network that can upscale an image, as there are other methods such as convolutional neural networks (CNN), specifically SRCNN. Upscaling an image is also known as Single Image Super-Resolution (SISR). A GAN model for upscaling images is called a Super-Resolution Generative Adversarial Network (SRGAN).

The goal for a GAN is to reconstruct a high-resolution image from a low-resolution input. What makes GANs different to SRCNNs is that SRCNNs assume an ideal bicubic down sampling kernel, which is different from real degradations where all sorts of noise, artifacts, compression, etc. come to play. A super-resolution image I^{SR} is reconstructed from a low-resolution image I^{LR}, which is a low-resolution version of its high-resolution image I^{HR}. I^{HR} is only available during training. In other words, during training there are two datasets: low-resolution images and high-resolution or ground truth images. The aim is to train a GAN model to be able to reconstruct an image with similar image quality to the ground truth images.

There are two neural networks in a GAN model: the generator and discriminator. A generating function G is trained to estimate for a given low-resolution input image its corresponding high-resolution or ground truth counterpart. A generator network like a feed-forward CNN G_{θ_G} parameterized with θ_G. $\theta_G = \{W_{1:L};b_{1:L}\}$ which denotes the weights and biases of an L-layer deep network and is obtained by optimizing an SR-specific loss function l^{SR}.

For training images $I_n^{HR}, n = 1, \dots, N$ with corresponding $I_n^{LR}, n = 1, \dots, N$, the following equation is solved:

$$\hat{\theta}_G = arg \min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR}) \quad (1)$$

where l^{SR} is the perceptual loss function.

A discriminator network D_{θ_D} is defined, in which to solve the adversarial min-max problem:

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{train}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{HR})))] \quad (2)$$

It allows one to train a generative model G with the goal of fooling a differentiable discriminator D that is trained to distinguish super-resolution images from ground truth images, which in turn allows the generator to create solutions that are similar to ground truth and thus difficult to classify by D. This is in contrast to solutions obtained by minimizing pixel-wise error measurements such as the MSE. This is the basic implementation of an SRGAN model.

There are various enhancements to the SRGAN model, namely ESRGAN (Enhanced SRGAN) and Real-ESRGAN, which in itself is an improvement to ESRGAN. ESRGAN improves upon SRGAN by introducing the residual-in-residual dense blocks (RRDB) to make the model more powerful and easier to train, and removing batch normalization layers which were used in SRGAN. The discriminator is replaced with a relativistic average GAN (RaGAN) to judge whether one image is more realistic than the other as opposed to classifying which image is real or fake. Perceptual loss is improved by using VGG features *before* activation instead of *after* as in SRGAN.

III. Implementation and Results

In this paper, the implementation that will be used is Real-ESRGAN, which is an extension of the ESRGAN model trained on pure synthetic data. A high-order degradation modeling process is introduced to better simulate real-world degradations. It uses the ESRGAN generator which is a deep network with multiple residual-in-residual dense blocks and a U-Net discriminator with spectral normalization.

The RRDB network architecture used by Real-ESRGAN is as follows:

```
class ResidualDenseBlock(nn.Module):
    def __init__(self, num_feat=64,
num_grow_ch=32):
        super(ResidualDenseBlock,
self).__init__()
        self.conv1 = nn.Conv2d(num_feat,
num_grow_ch, 3, 1, 1)
        self.conv2 = nn.Conv2d(num_feat +
num_grow_ch, num_grow_ch, 3, 1, 1)
        self.conv3 = nn.Conv2d(num_feat + 2
* num_grow_ch, num_grow_ch, 3, 1, 1)
        self.conv4 = nn.Conv2d(num_feat + 3
* num_grow_ch, num_grow_ch, 3, 1, 1)
        self.conv5 = nn.Conv2d(num_feat + 4
* num_grow_ch, num_feat, 3, 1, 1)

        self.lrelu =
```

```
nn.LeakyReLU(negative_slope=0.2,
inplace=True)

        default_init_weights([self.conv1,
self.conv2, self.conv3, self.conv4,
self.conv5], 0.1)

    def forward(self, x):
        x1 = self.lrelu(self.conv1(x))
        x2 =
self.lrelu(self.conv2(torch.cat((x, x1),
1)))
        x3 =
self.lrelu(self.conv3(torch.cat((x, x1, x2),
1)))
        x4 =
self.lrelu(self.conv4(torch.cat((x, x1, x2,
x3), 1)))
        x5 = self.conv5(torch.cat((x, x1,
x2, x3, x4), 1))
        return x5 * 0.2 + x


class RRDB(nn.Module):
    def __init__(self, num_feat,
num_grow_ch=32):
        super(RRDB, self).__init__()
        self.rdb1 =
ResidualDenseBlock(num_feat, num_grow_ch)
        self.rdb2 =
ResidualDenseBlock(num_feat, num_grow_ch)
        self.rdb3 =
ResidualDenseBlock(num_feat, num_grow_ch)

    def forward(self, x):
        out = self.rdb1(x)
        out = self.rdb2(out)
        out = self.rdb3(out)
        return out * 0.2 + x


class RRDBNet(nn.Module):
    def __init__(self, num_in_ch,
num_out_ch, scale=4, num_feat=64,
num_block=23, num_grow_ch=32):
        super(RRDBNet, self).__init__()
        self.scale = scale
        if scale == 2:
            num_in_ch = num_in_ch * 4
        elif scale == 1:
            num_in_ch = num_in_ch * 16
        self.conv_first =
nn.Conv2d(num_in_ch, num_feat, 3, 1, 1)
        self.body = make_layer(RRDB,
num_block, num_feat=num_feat,
num_grow_ch=num_grow_ch)
        self.conv_body = nn.Conv2d(num_feat,
num_feat, 3, 1, 1)
        # upsample
        self.conv_up1 = nn.Conv2d(num_feat,
num_feat, 3, 1, 1)
        self.conv_up2 = nn.Conv2d(num_feat,
num_feat, 3, 1, 1)
        self.conv_hr = nn.Conv2d(num_feat,
num_feat, 3, 1, 1)
        self.conv_last = nn.Conv2d(num_feat,
num_out_ch, 3, 1, 1)

        self.lrelu =
nn.LeakyReLU(negative_slope=0.2,
inplace=True)

    def forward(self, x):
        if self.scale == 2:
            feat = pixel_unshuffle(x,
scale=2)
        elif self.scale == 1:
            feat = pixel_unshuffle(x,
scale=4)
        else:
            feat = x
        feat = self.conv_first(feat)
        body_feat =
self.conv_body(self.body(feat))
        feat = feat + body_feat
        # upsample
        feat =
self.lrelu(self.conv_up1(F.interpolate(feat,
scale_factor=2, mode='nearest')))
        feat =
self.lrelu(self.conv_up2(F.interpolate(feat,
scale_factor=2, mode='nearest')))
        out =
self.conv_last(self.lrelu(self.conv_hr(feat)
))
        return out
```

This RRDB generator network can be used to do 2x upscaling and 4x upscaling on each dimension (4x and 16x resolution respectively).

The U-Net discriminator network used by Real-ESRGAN is as follows:

```
class UNetDiscriminatorSN(nn.Module):
    def __init__(self, num_in_ch,
num_feat=64, skip_connection=True):
        super(UNetDiscriminatorSN,
self).__init__()
        self.skip_connection =
skip_connection
        norm = spectral_norm
        self.conv0 = nn.Conv2d(num_in_ch,
num_feat, kernel_size=3, stride=1,
padding=1)
        # downsample
        self.conv1 =
norm(nn.Conv2d(num_feat, num_feat * 2, 4, 2,
1, bias=False))
        self.conv2 = norm(nn.Conv2d(num_feat
* 2, num_feat * 4, 4, 2, 1, bias=False))
        self.conv3 = norm(nn.Conv2d(num_feat
* 4, num_feat * 8, 4, 2, 1, bias=False))
        # upsample
        self.conv4 = norm(nn.Conv2d(num_feat
* 8, num_feat * 4, 3, 1, 1, bias=False))
        self.conv5 = norm(nn.Conv2d(num_feat
* 4, num_feat * 2, 3, 1, 1, bias=False))
```

```
        self.conv6 = norm(nn.Conv2d(num_feat
* 2, num_feat, 3, 1, 1, bias=False))
        # extra convolutions
        self.conv7 =
norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=False))
        self.conv8 =
norm(nn.Conv2d(num_feat, num_feat, 3, 1, 1,
bias=False))
        self.conv9 = nn.Conv2d(num_feat, 1,
3, 1, 1)

    def forward(self, x):
        # downsample
        x0 = F.leaky_relu(self.conv0(x),
negative_slope=0.2, inplace=True)
        x1 = F.leaky_relu(self.conv1(x0),
negative_slope=0.2, inplace=True)
        x2 = F.leaky_relu(self.conv2(x1),
negative_slope=0.2, inplace=True)
        x3 = F.leaky_relu(self.conv3(x2),
negative_slope=0.2, inplace=True)

        # upsample
        x3 = F.interpolate(x3,
scale_factor=2, mode='bilinear',
align_corners=False)
        x4 = F.leaky_relu(self.conv4(x3),
negative_slope=0.2, inplace=True)

        if self.skip_connection:
            x4 = x4 + x2
        x4 = F.interpolate(x4,
scale_factor=2, mode='bilinear',
align_corners=False)
        x5 = F.leaky_relu(self.conv5(x4),
negative_slope=0.2, inplace=True)

        if self.skip_connection:
            x5 = x5 + x1
        x5 = F.interpolate(x5,
scale_factor=2, mode='bilinear',
align_corners=False)
        x6 = F.leaky_relu(self.conv6(x5),
negative_slope=0.2, inplace=True)

        if self.skip_connection:
            x6 = x6 + x0

        out = F.leaky_relu(self.conv7(x6),
negative_slope=0.2, inplace=True)
        out = F.leaky_relu(self.conv8(out),
negative_slope=0.2, inplace=True)
        out = self.conv9(out)

        return out
```

Before inference, the image needs to be preprocessed. The image is padded if needed before feeding into the neural network to make sure it is divisible.

```
def pre_process(self, img):
    img = torch.from_numpy(np.transpose(img,
(2, 0, 1))).float()

self.img = img.unsqueeze(0).to(self.device)
    if self.half:
        self.img = self.img.half()
    # pre_pad
    if self.pre_pad != 0:
        self.img = F.pad(self.img, (0,
self.pre_pad, 0, self.pre_pad), 'reflect')
```

After feeding into the neural network, the resulting image is post-processed. The padding is removed.

```
def post_process(self):
    # remove prepad
    if self.pre_pad != 0:
        _, _, h, w = self.output.size()
        self.output = self.output[:, :, 0:h
- self.pre_pad * self.scale, 0:w -
self.pre_pad * self.scale]
    return self.output
```

Figure 2 Top: original image, bottom: upscaled image by Real-ESRGAN

The original image is a low-quality, low-resolution image with just 500x331 pixels, while the right is noticeably sharper with a resolution of 2000x1404, a 4x increase in each dimension or a 16x increase in total. Some fringing and minor artifacts are still visible as a result of upscaling.

Figure 3 Top: original image, bottom: upscaled image

In this example, the artifacts are less visible while the details are once again noticeably sharper. The better the source image, the higher quality the resulting upscaled image is. That applies to any method of image upscaling. The source image has a resolution of 899x418 while the upscaled image has a resolution of 3596x1672.

Figure 4 Top: original image, bottom: upscaled image

Once again, the image is vastly sharper than the source image. JPEG compression artifacts which are visible in the entirety of the source image are gone in the resulting image, but there are some artifacts around the license plate and fringing in the trees in the background. The details on the source image are insufficient for the neural network to upscale the image. Hence, the resulting image looks sharp with defined edges on the car but the neural network has free rein on the license plate, as it has no general idea what the original text could have been. This "limitation" is extremely hard to overcome for any artificial neural network, as the image lacks enough information for the network.

The source image's resolution is 517x352, while the upscaled image is 2068x1408.

IV. Conclusion

In conclusion, generative adversarial networks can be used to increase both image resolution and image quality. While both don't always go hand in hand (images can have higher quality with lower resolutions), generative adversarial networks improve on both with minimal artifacts resulting from model inference. Of course, like any other solution, there are limitations in which the artificial neural network simply cannot overcome, as is the example in Figure 4. With artificial neural networks capable of upscaling images, restoring old images can be done without much hassle.

Acknowledgment

This paper would not have been brought to fruition without resources provided the IF4073 Digital Image Processing class with the guidance of Dr. Ir. Rinaldi, M.T as the lecturer of the author in class.

The author would like to thank other brilliant programmers and researchers in the world for sharing their knowledge and findings, without it this paper would not have been complete.

References

[1] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A. et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," CVPR, 2017.

[2] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., et al. "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks," ECCV, 2018.

[3] Wang, X., Xie, L., Dong, C., Shan, Y., "Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data," Applied Research Center Tencent PCG, Shenzen Institutes of Advanced Technology, University of Chinese Academy of Sciences, 2021.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Januari 2025

Ammar Rasyad Chaeroel 13521136