**Part-1**

SRGAN, introduced in the paper "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," revolutionizes the field of image super-resolution by employing a deep Generative Adversarial Network (GAN). The core idea is to upscale low-resolution images into high-resolution counterparts, not just with pixel accuracy, but with an emphasis on perceptual quality.

The architecture consists of two primary components: a deep residual network (ResNet) based generator and a discriminator network. The generator, responsible for the upscaling task, is designed with deep convolutional layers and residual blocks that facilitate the flow of gradients during training, enabling the effective training of deeper networks. The discriminator's role is to distinguish between the super-resolved (generated) images and the original high-resolution images, adding an adversarial aspect to the training process.

A main idea in this paper of SRGAN is the introduction of **perceptual loss**, which differs from traditional mean squared error (MSE) based losses. This perceptual loss is computed using feature maps from a pre-trained VGG network and is instrumental in ensuring that the super-resolved images are not just pixel-accurate but also possess high fidelity in terms of texture and finer details. This approach allows SRGAN to generate images that are more photo-realistic compared to those produced by previous methods focused solely on optimizing pixel-wise accuracy.

A diagram of a sequence of steps

Description automatically generated

Picture reference from GeeksforGeeks

A diagram of a computer network

Description automatically generated with medium confidence

The above picture credit: medium.

This is what we have done while training our SRGAN:

For the core of the SRGAN, we have define a Generator and a Discriminator model. The Generator is tasked with upscaling low-resolution images, employing convolutional layers, residual blocks, and upsampling blocks to progressively enhance image details. The Discriminator, on the other hand, works to distinguish between upscaled images and original high-resolution images, driving the adversarial training process. You also implement a perceptual loss function using features extracted from a pre-trained VGG network, focusing on achieving high perceptual quality in the upscaled images.

The training process is encapsulated within the train\_fn function, where both the Generator and Discriminator are trained in a coordinated manner, using a combination of content loss (MSE) and adversarial loss. For Visualization, we have defined the functions where inspection of the model's performance is facilitated through the plot\_examples and plot\_generated functions, the former visualizing images from the training dataset and the latter showcasing the model's upscaling ability on the test dataset.

The training loop iterates over a set number of epochs, displaying processed images, calculating losses, updating model parameters, and saving model checkpoints at specified intervals.

1. Transformations:

- transform\_low and transform\_high are transformation pipelines defined using PyTorch's `transforms` module. They preprocess images by resizing and converting them to tensors. Low-resolution images are resized to 25x25, and high-resolution images to 100x100.

2. Custom Dataset Class (ImageDataset):

It's designed to handle loading of low and high-resolution images from a specified directory .The class uses the provided transformations for preprocessing.

3. Single Image Dataset (SingleImageDataset):

- Another custom dataset class that handles individual images of varying sizes in a specified directory, Test, which contains the image files of different sizes. It applies a defined transformation to each image.

4. Models:

- Generator: A neural network designed to upscale low-resolution images to high-resolution images.

- Discriminator: A neural network that differentiates between real high-resolution images and upscaled images produced by the generator.

-Perceptual Loss (VGG-based Loss): A loss function that compares the feature representations of the upscaled and high-resolution images. Utilizes a pre-trained VGG19 network, specifically the first 25 layers, to extract feature representation

Lets look into the Generator model architecture has the following layers

-Initial Convolution Block, Residual Blocks, Up sampling Blocks, Final Convolution Layer

The core of the Generator is made up of multiple residual blocks. Each block has two Convolution Blocks. Each Convolution Block within the residual block consists of a convolutional layer , the output of the second Convolution Block in a residual block is added to its input, forming a skip connection that helps mitigate the vanishing gradient problem and allows for deeper network architectures.

Following the residual blocks, the network includes upsampling blocks. Each upsampling block uses a Conv2d layer to increase the number of channels, followed by PixelShuffle for upsampling and a PReLU activation. The upsampling process is repeated to progressively increase the spatial dimensions of the feature maps to reach the desired high-resolution output size.

Now , the Discriminator architecture is implemented in the code:

Convolutional Blocks: Series of ConvBlocks with increasing number of features. Stride alternates to reduce spatial dimensions.

Adaptive Pooling and Flatten: Resizes feature maps to a fixed size and flattens them.

Linear Layers: Two linear layers process the flattened features to produce a single output, signifying whether the input is real or generated.

Forward Method: Executes the sequential processing of an input image to classify it as real or fake.

5. Training Function (train\_fn):

- A function that performs the training of the generator and discriminator. It includes forward passes, loss calculations, and backpropagation.

6. Image Display Functions : helps to plot the images

To conclude, here in the code, we have trained our GAN with the images present inside the HR,LR folders on multiple epochs , and tried to see how SRGAN produces the image using the different images.

This should produce a good quality image, but due to the platform constraint we could run for few epochs only.