Analytics Club, IITM

Super-Resolution Generative Adversarial Networks

SRGANs

What is Super-Resolution?

What is Super-Resolution?

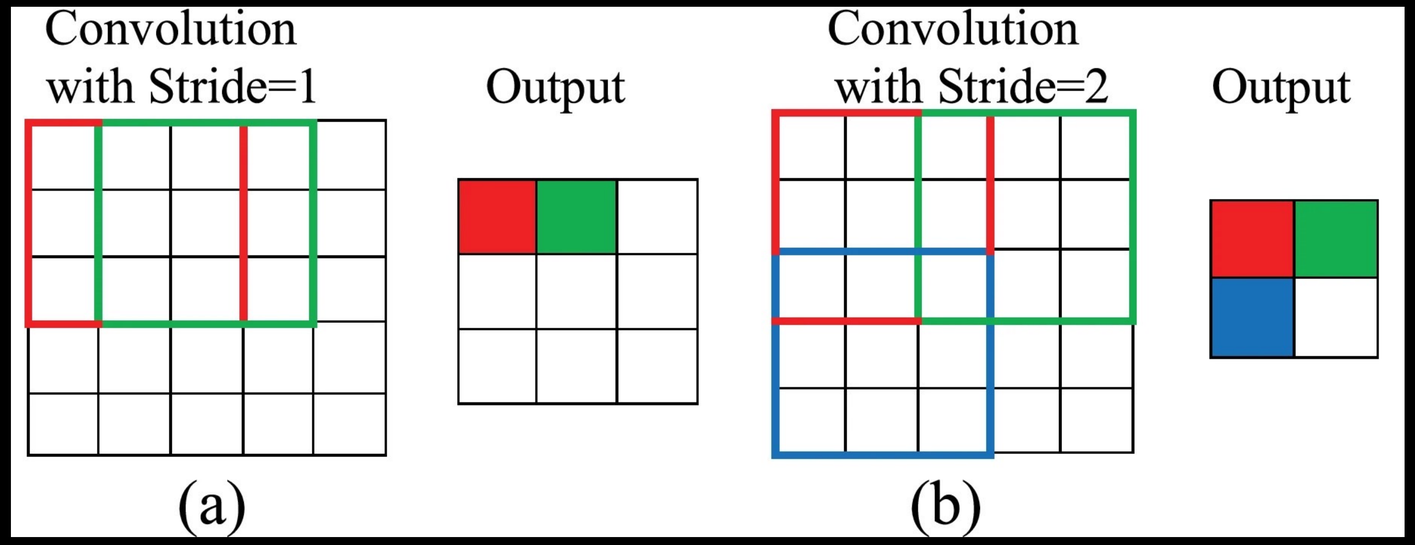
- Super-Resolution is the process of upscaling a low resolution image into a high resolution image.
- Why? Because high resolution images have a higher pixel density and hence have more detail
- Commonly used in medical imaging and surveillance.

Today, we look at a GAN based architecture for performing super-resolution,
SRGANs,
which can perform upto 4x upscaling

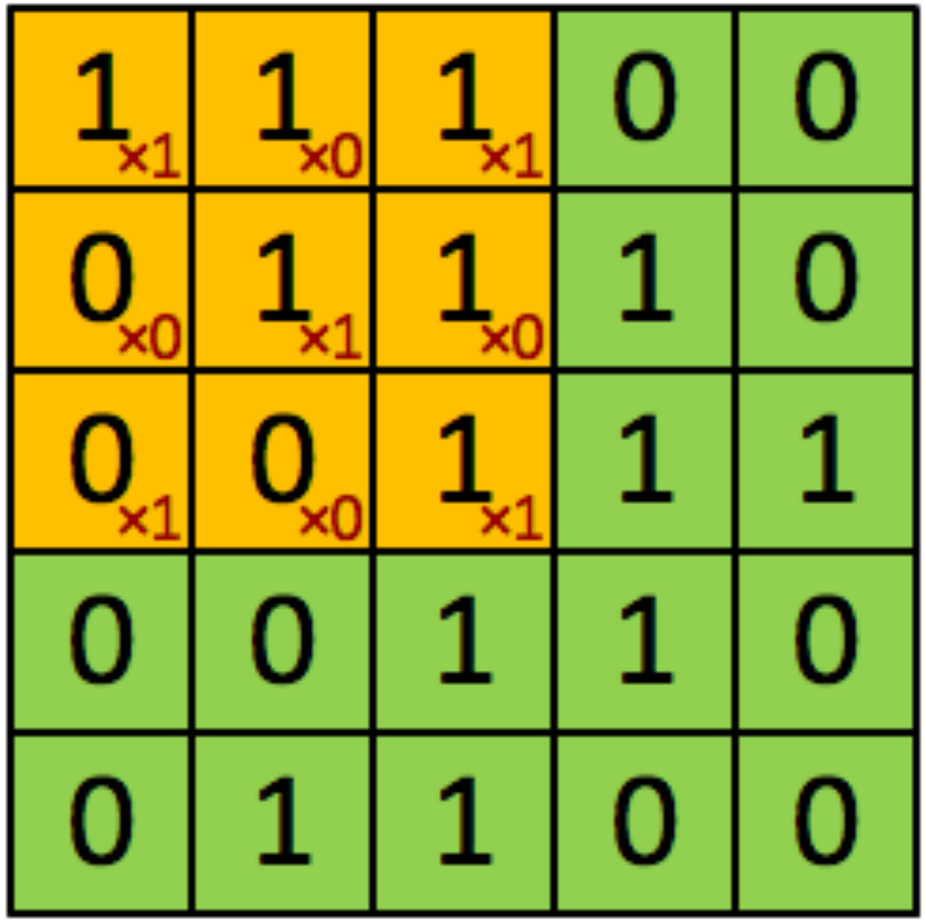
Some quick concepts

Kernels and Stride

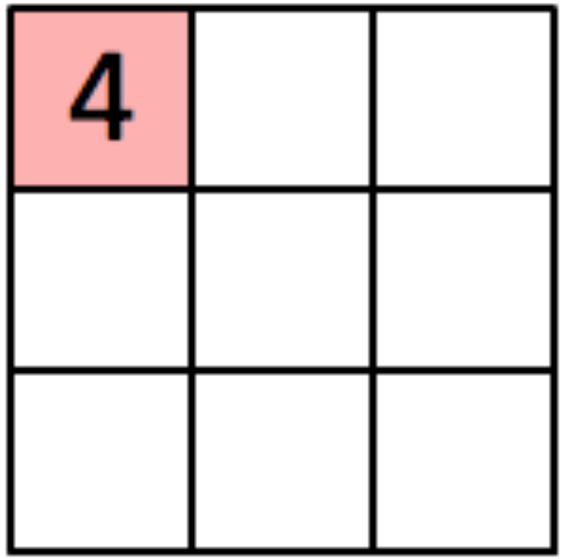
- Pick a matrix of weights and perform element-wise multiplication with the corresponding pixels of a small window of the image



- The chosen matrix is called a kernel
- The size of the matrix chosen is called the kernel size
- The number of pixels skipped between successive convolutions is called stride



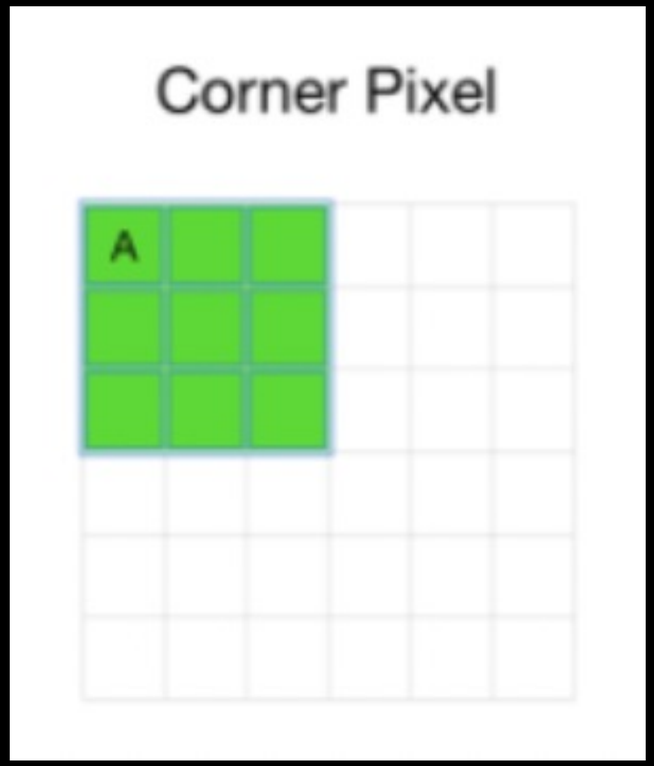
Image

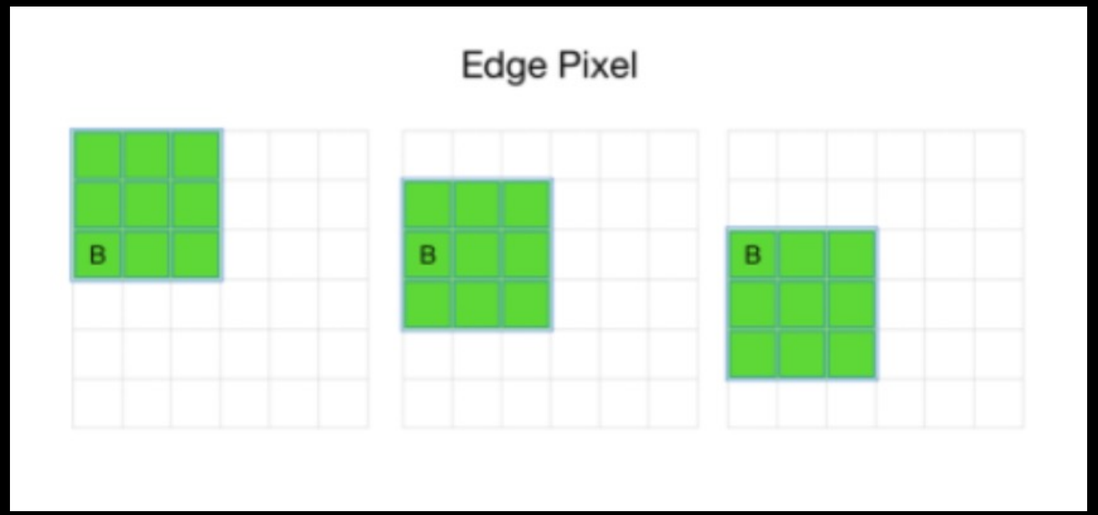


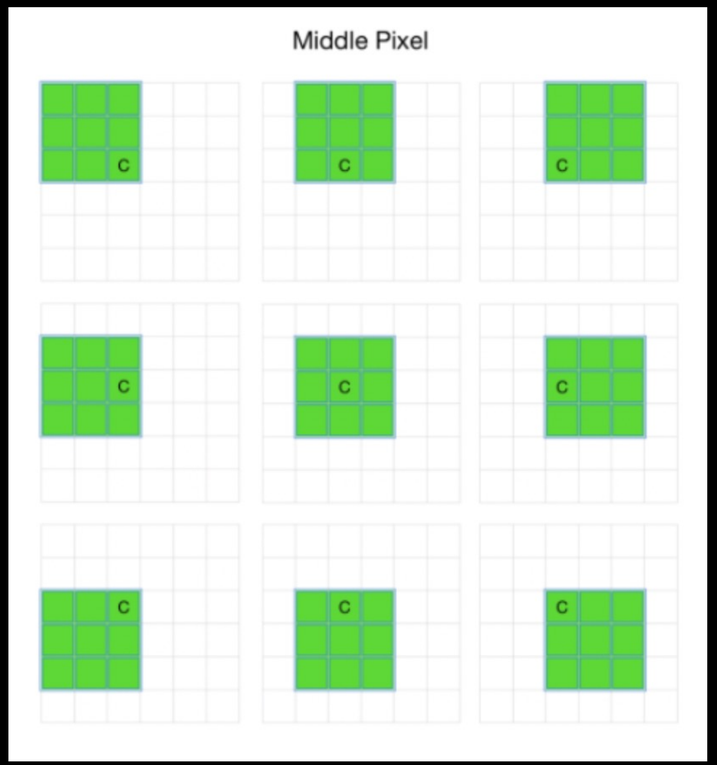
Convolved Feature

Padding

- When we perform a kernel operation, all pixels are not covered equally

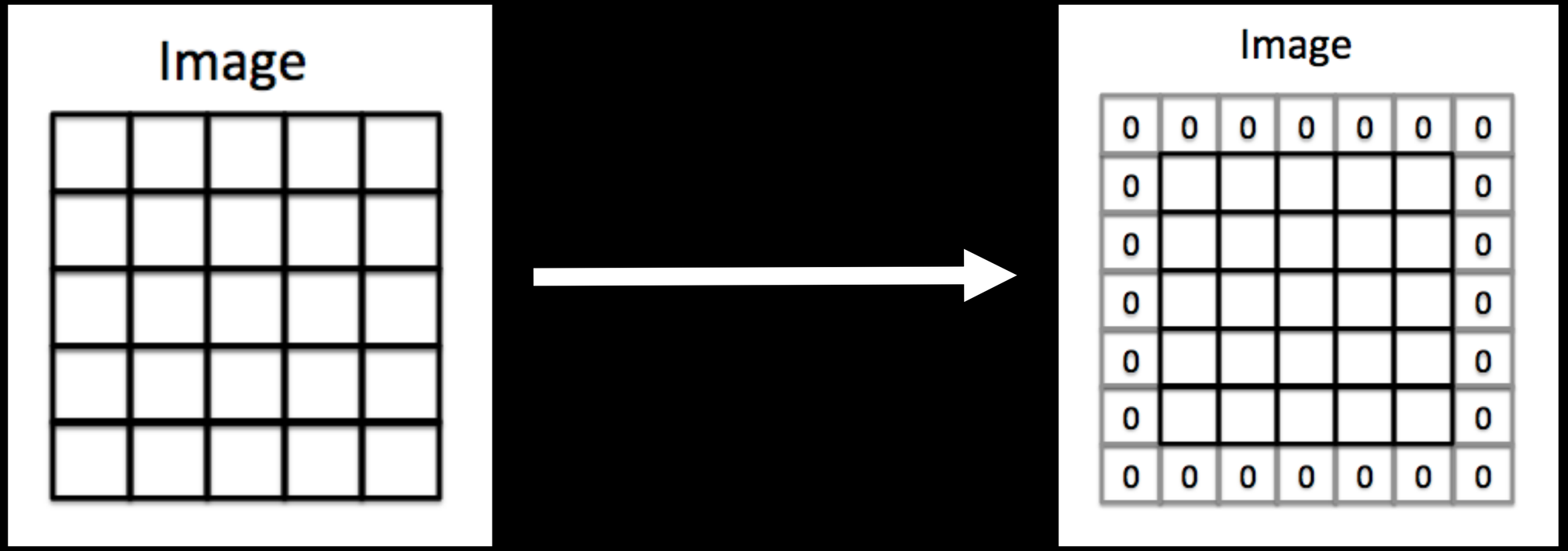






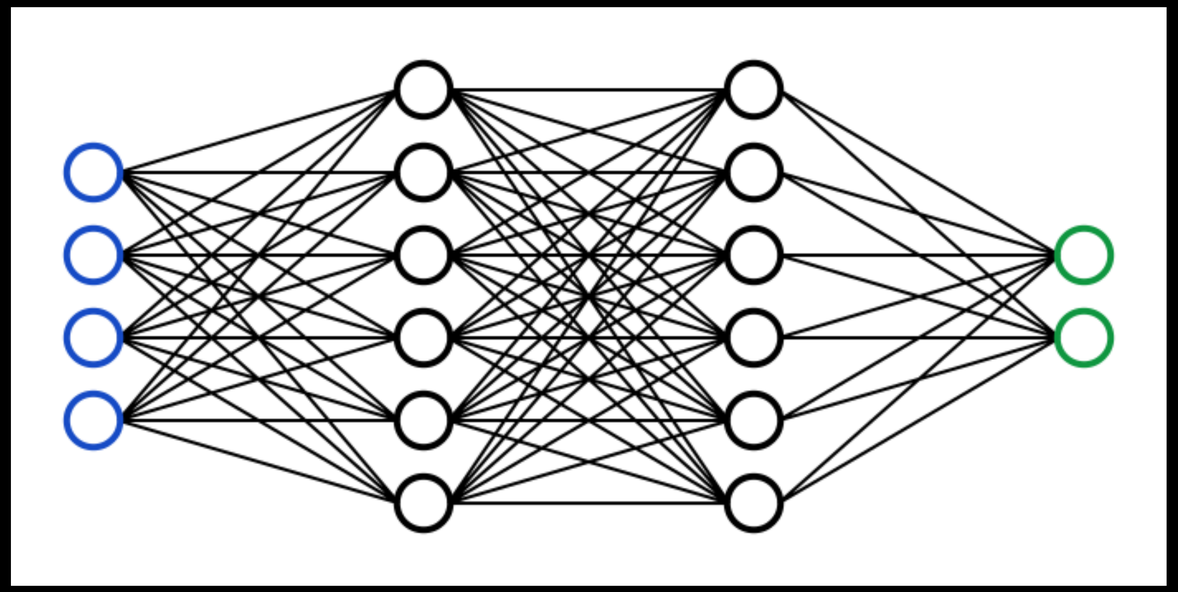
Padding

- To solve these problems, we apply padding to our image



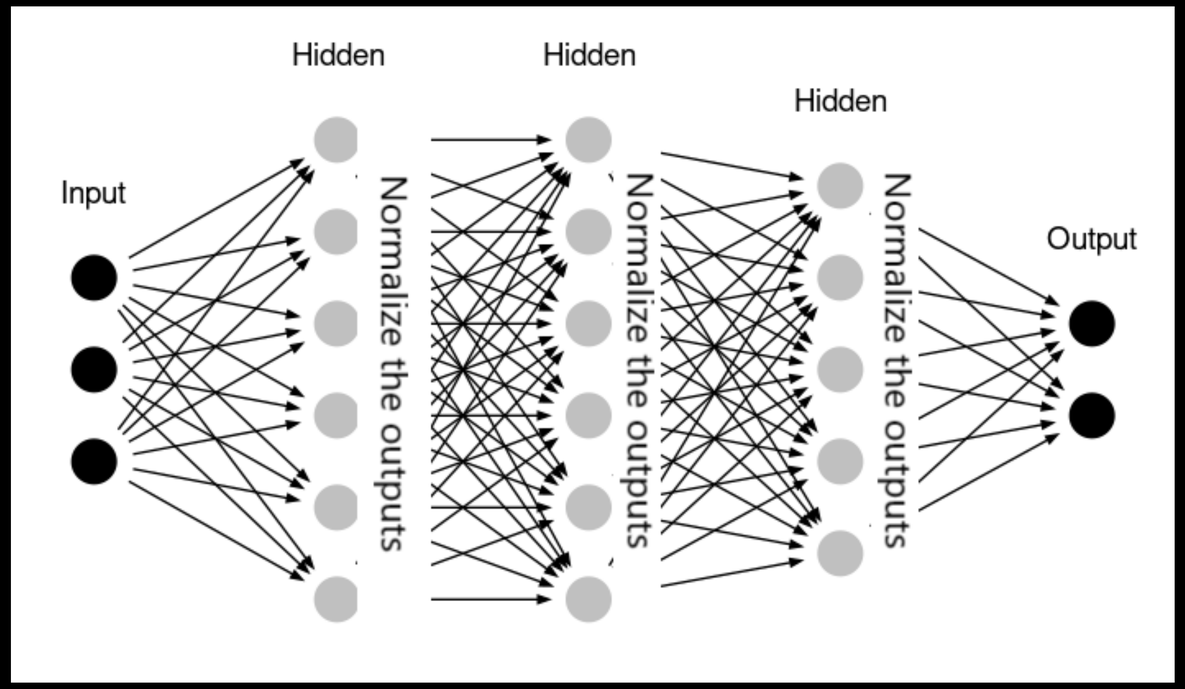
- One particular type of padding is “Same Padding” where we add padding such that the output features have same dimensions as the input features

Batch Normalization



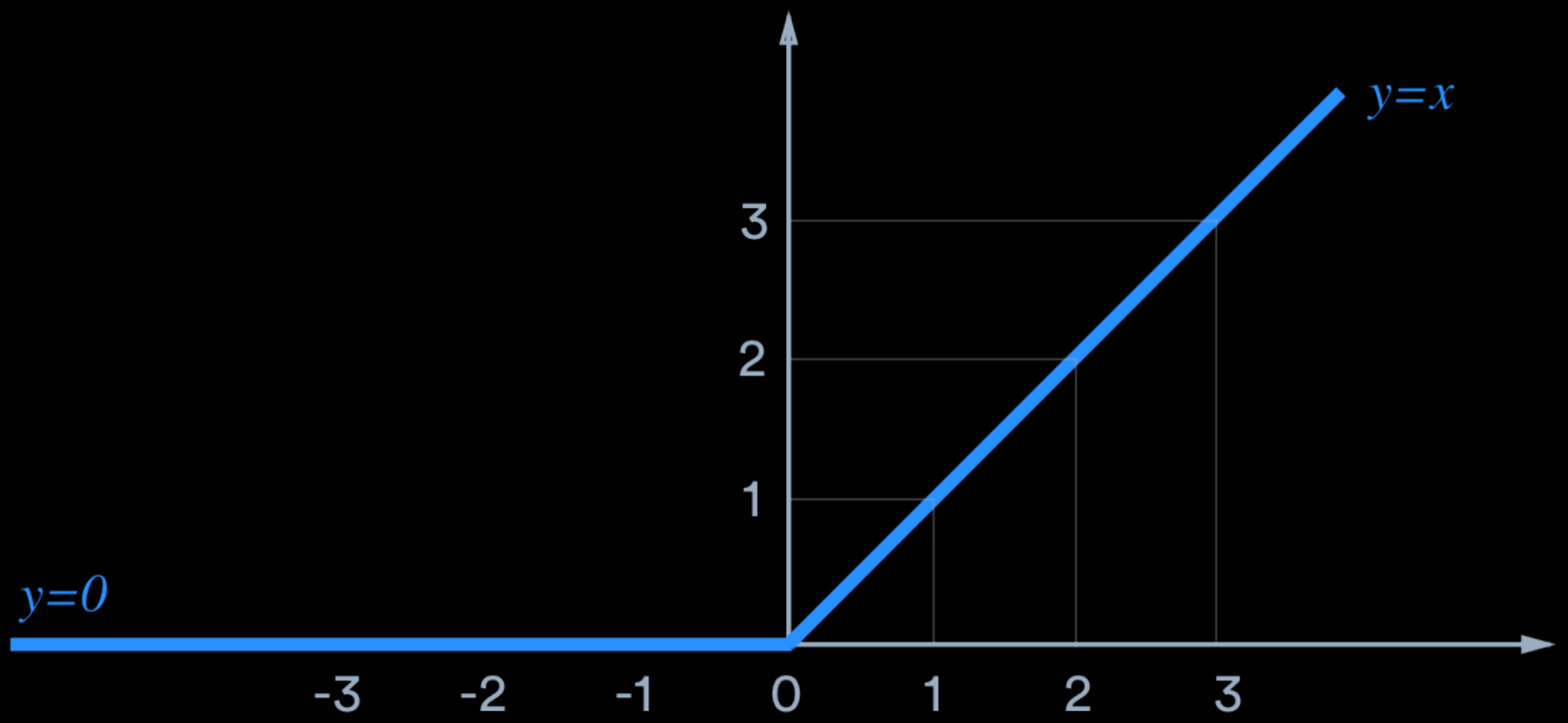
- As we train the model, the weights and biases of the neurons may vary greatly in magnitude
- But, when performing back-propagation, we assume the parameters for the previous layers are the same
- The varying weights leads to a very unstable back-propagation
- The model is chasing a “moving target” in some sense
- “Internal Covariate Shift”

Batch Normalization



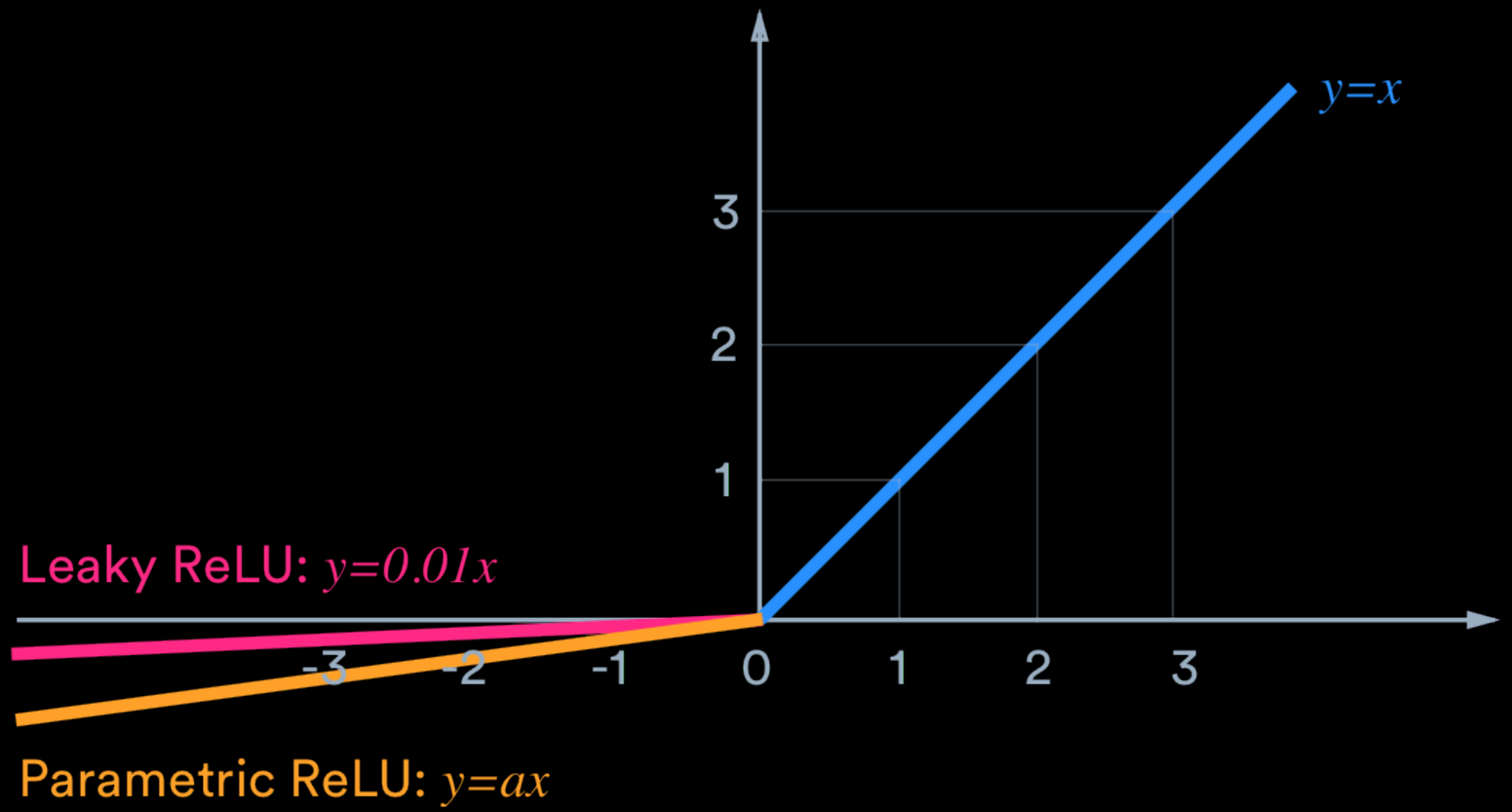
- To prevent this, we perform batch normalization
- We normalize the outputs of each layer
- The normalization can be done before or after the activation
- This improves training efficiency and has a regularization effect

Rectified Linear Unit(ReLU)



- When the activations are postive, they are let through as they are
- When they are negative, the output of the ReLU layer is 0

LeakyReLU and PReLU

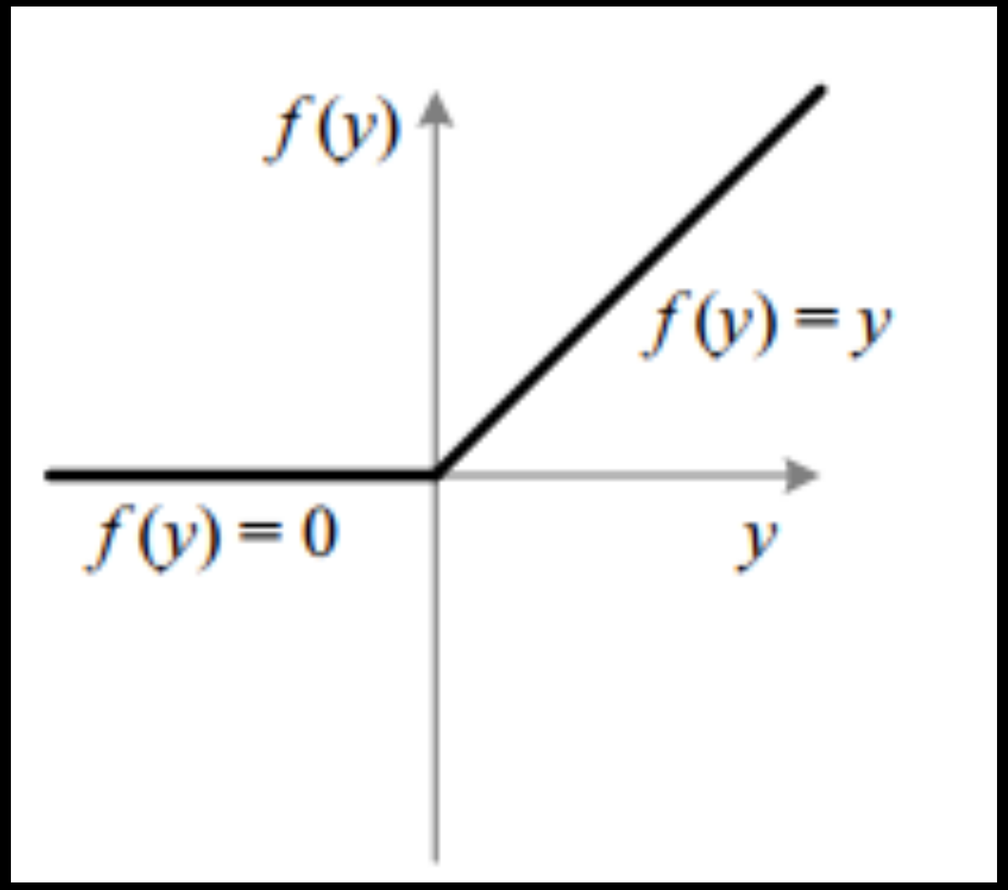


- In LeakyReLU, there is a small slope for negative values as well instead of zero.
- The slope is a hyper-parameter.
- In PReLU(Parametric ReLU), this slope is learnt by the network

Quick Question

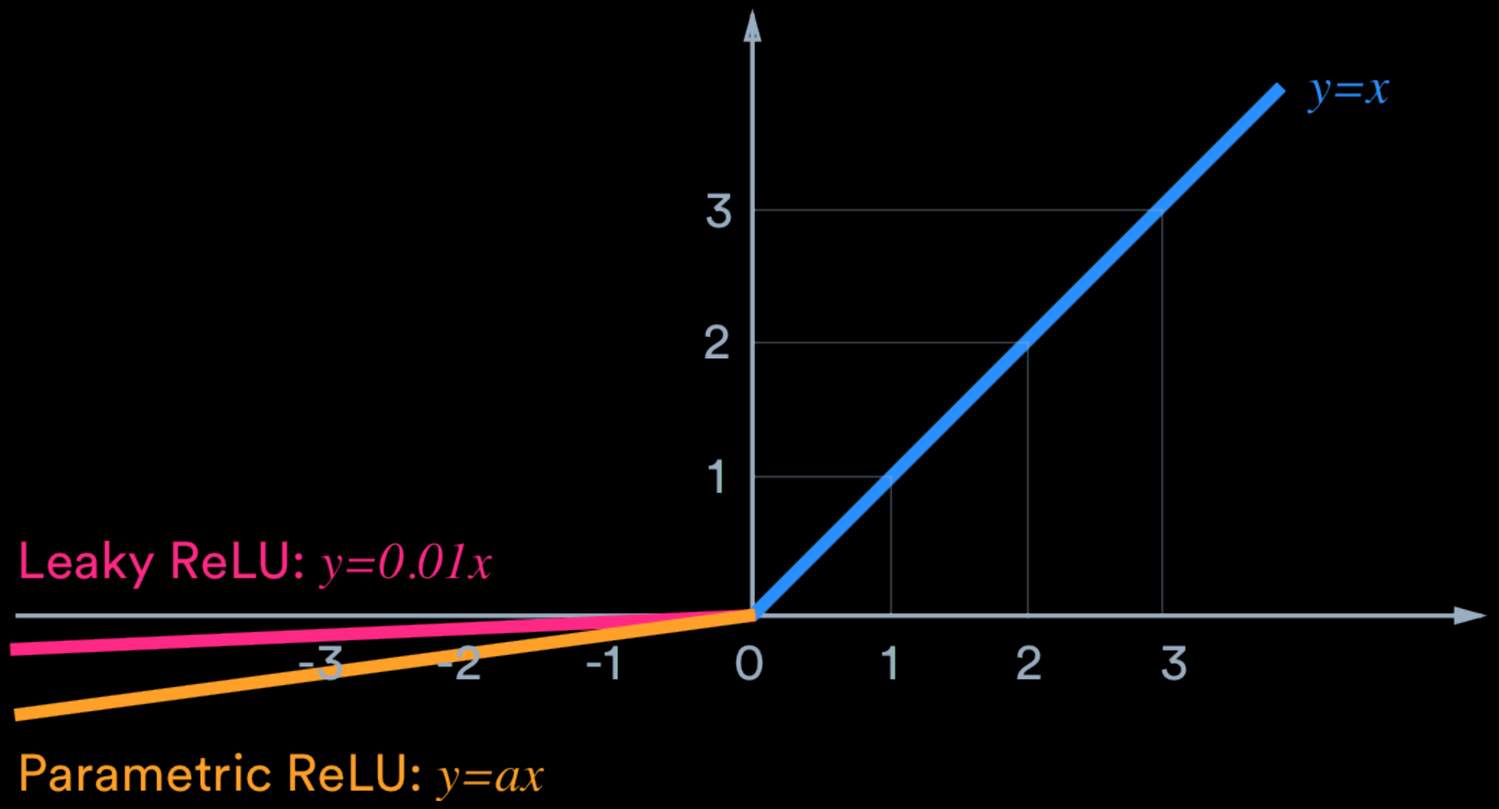
Why do we use
PReLU/LeakyReLU
over
Normal ReLU?

The Dead Neuron Problem



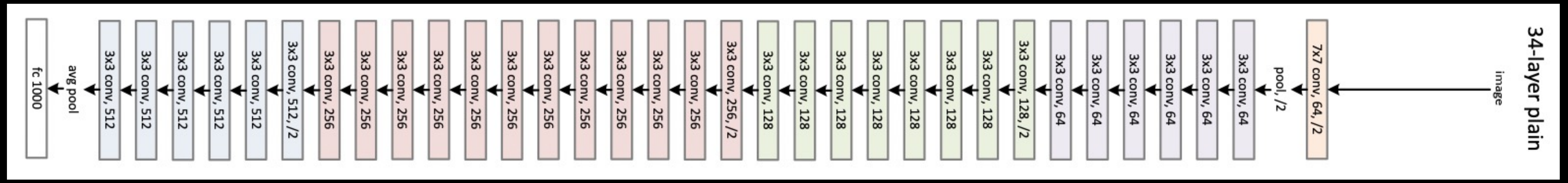
- The slope of ReLU in negative side is zero
- So, if a neuron outputs a negative value, it is very unlikely to change.
- Such neurons are said to be “dead” and will always output zero.
- Due to bad initialisation/high learning rate, the network might end up with a lot of dead neurons
- So a large part of the network might do nothing

The Dead Neuron Problem



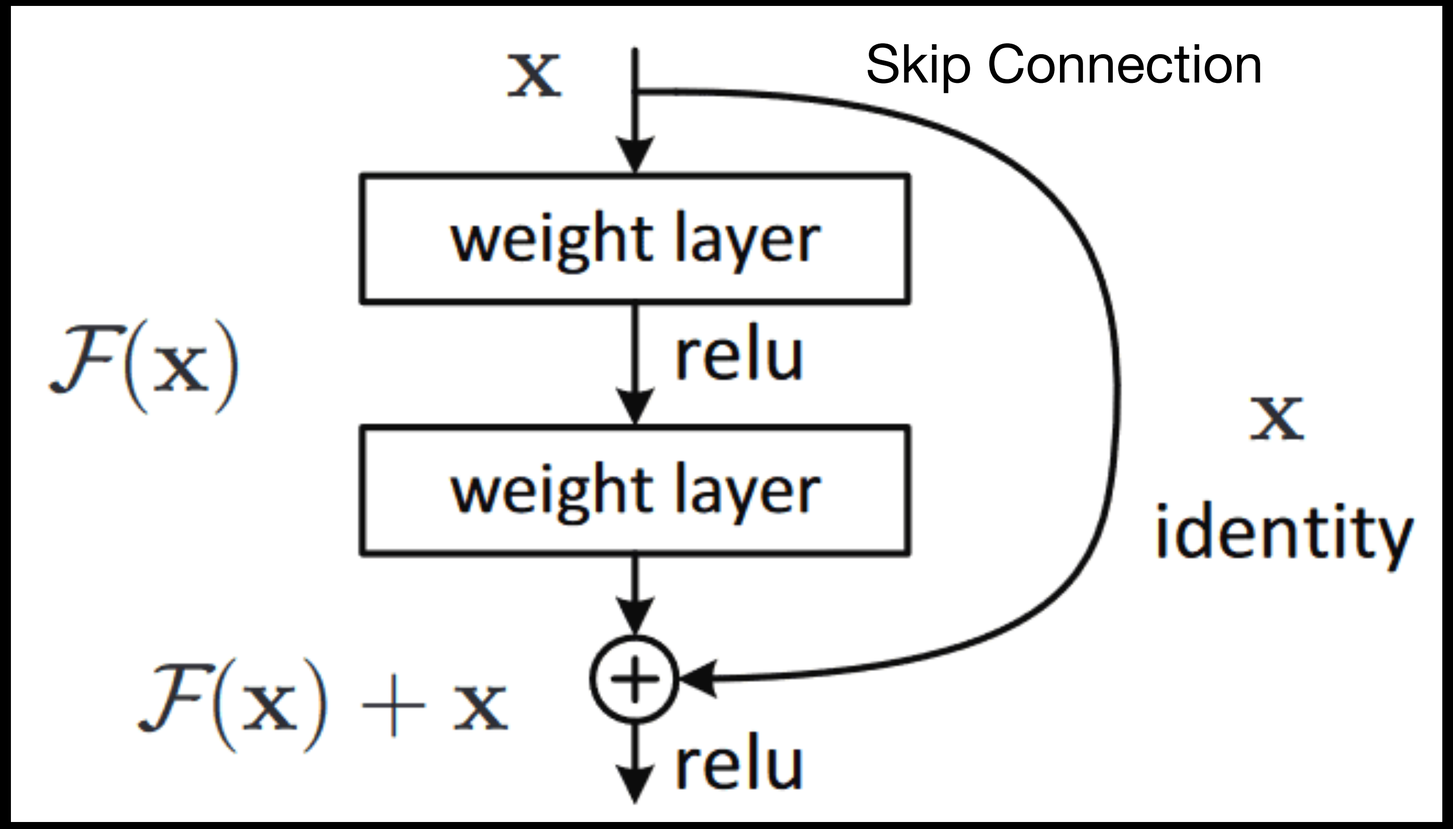
- To solve this problem, we use LeakyReLU or PReLU
- They prevent neruons becoming “dead” neurons by allowing a portion of the negative activation to pass through
- They improve gradient flow

Vanishing Gradient Problem



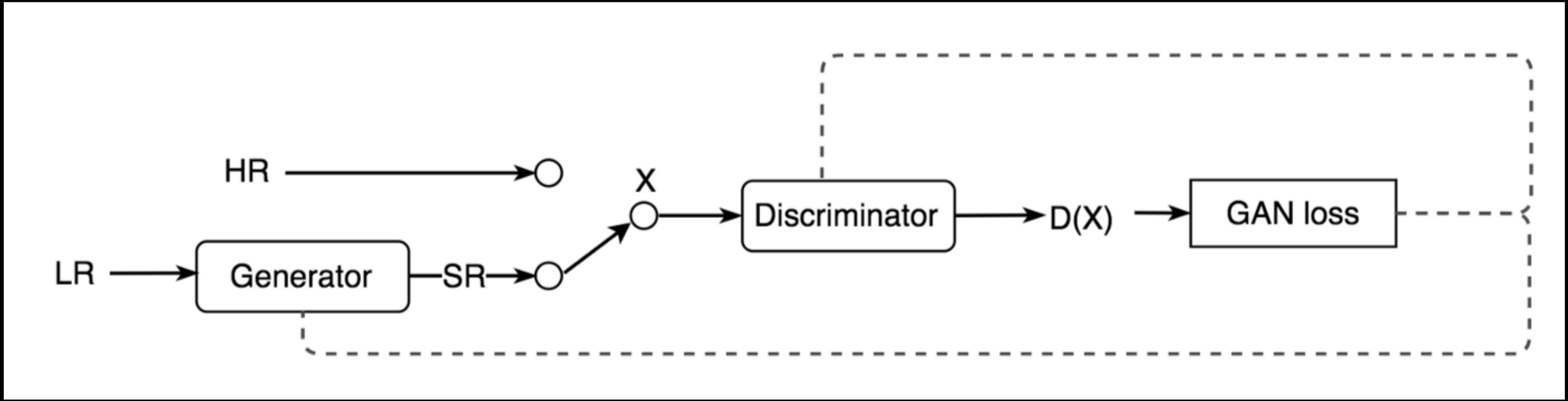
- In deep neural networks, at deeper levels it is likely that the gradient vanishes, i.e, becomes zero
- Hence, the model doesn’t improve

Skip-Connections



- Skip-connections allow an alternative path for gradient to follow and hence avoids the vanishing gradient problem.
- The network learns the identity function, which means the deeper layers perform atleast as well as the higher layers

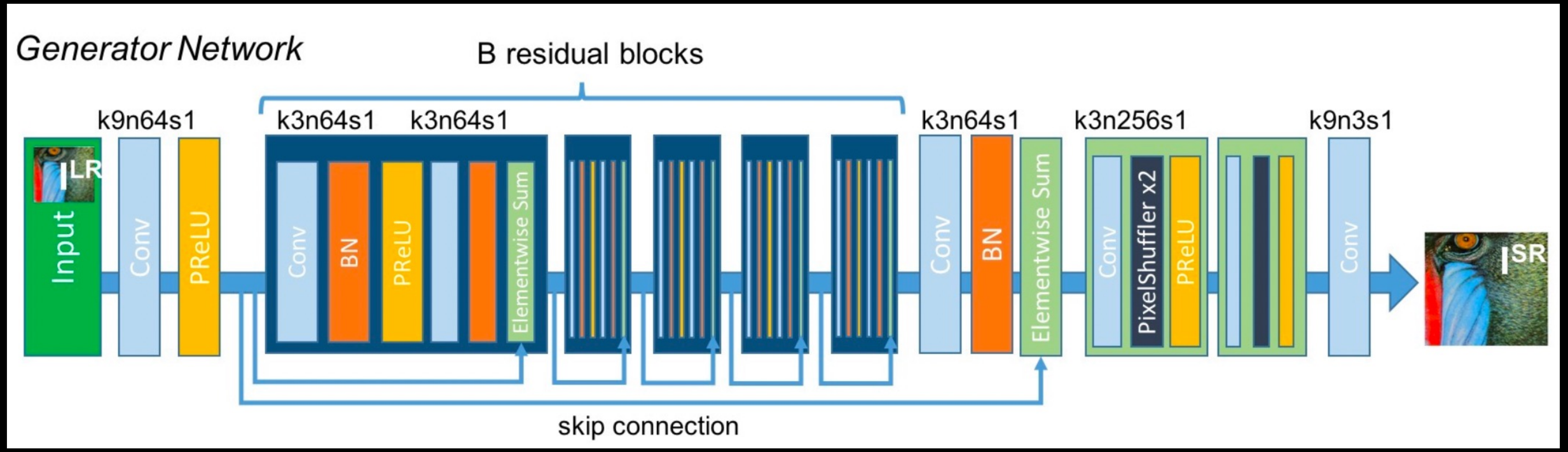
SRGAN Architecture



- A generator network
- A discriminator network
- A specialized GAN loss function

Generator Architecture

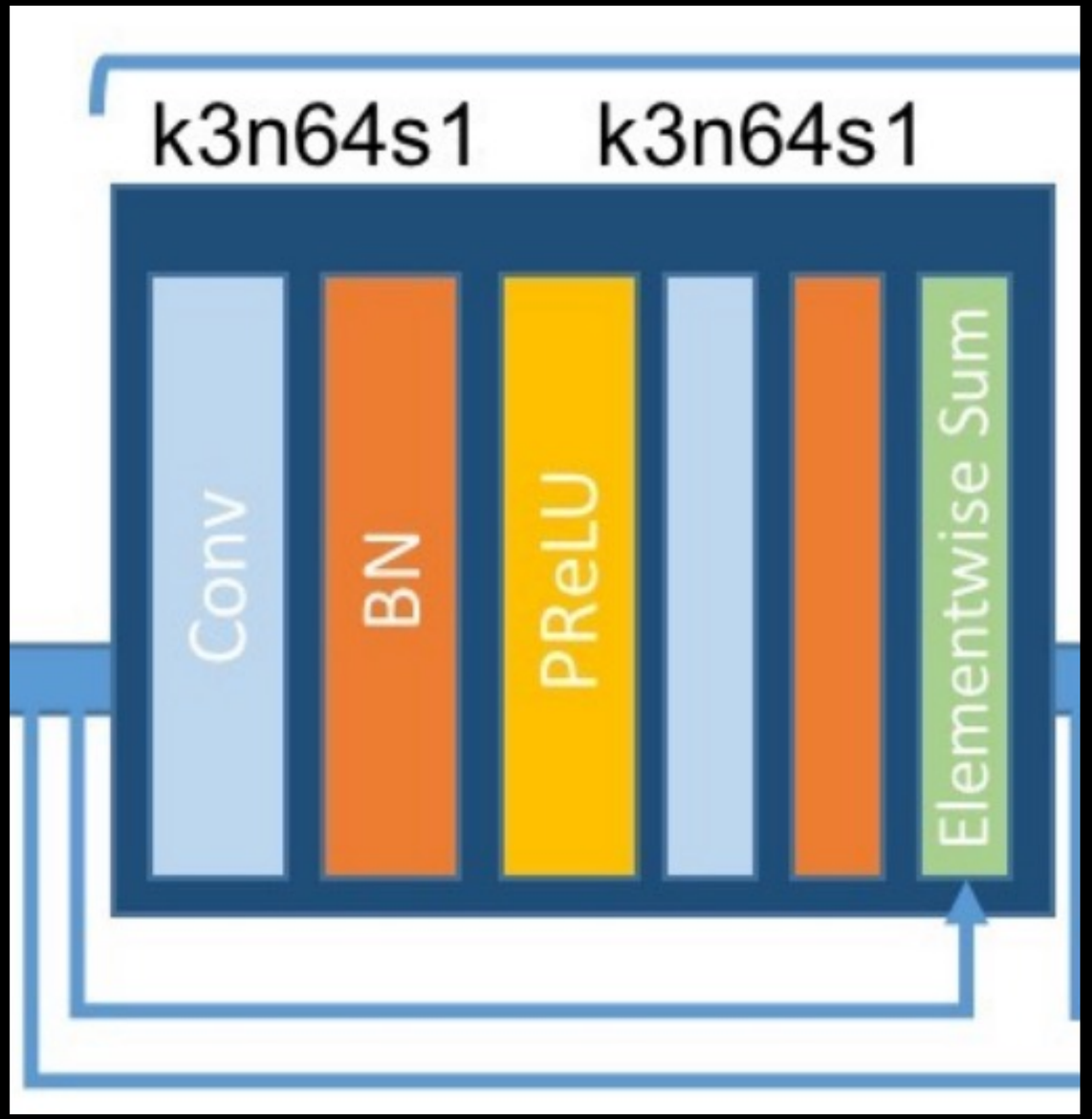
Generator Architecture



The generator consists of two major components:

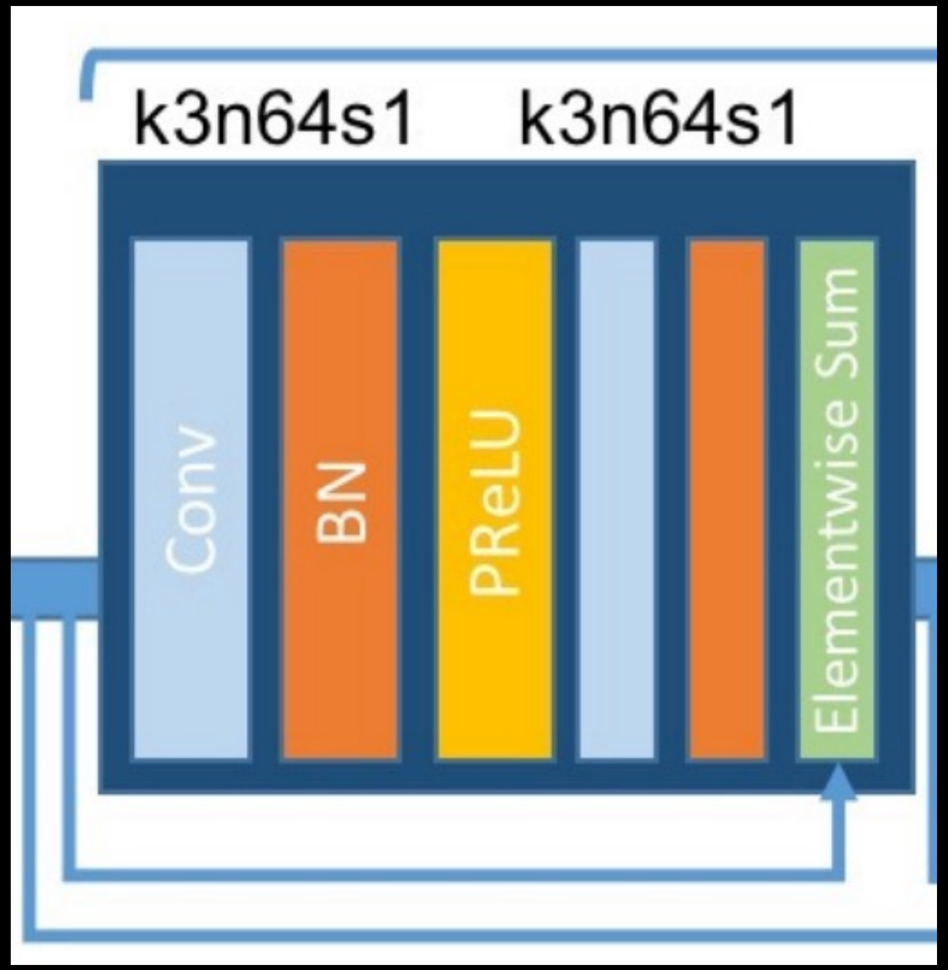
1. Residual block
2. Sub-Pixel Convolution(Upscaling)

Residual Blocks

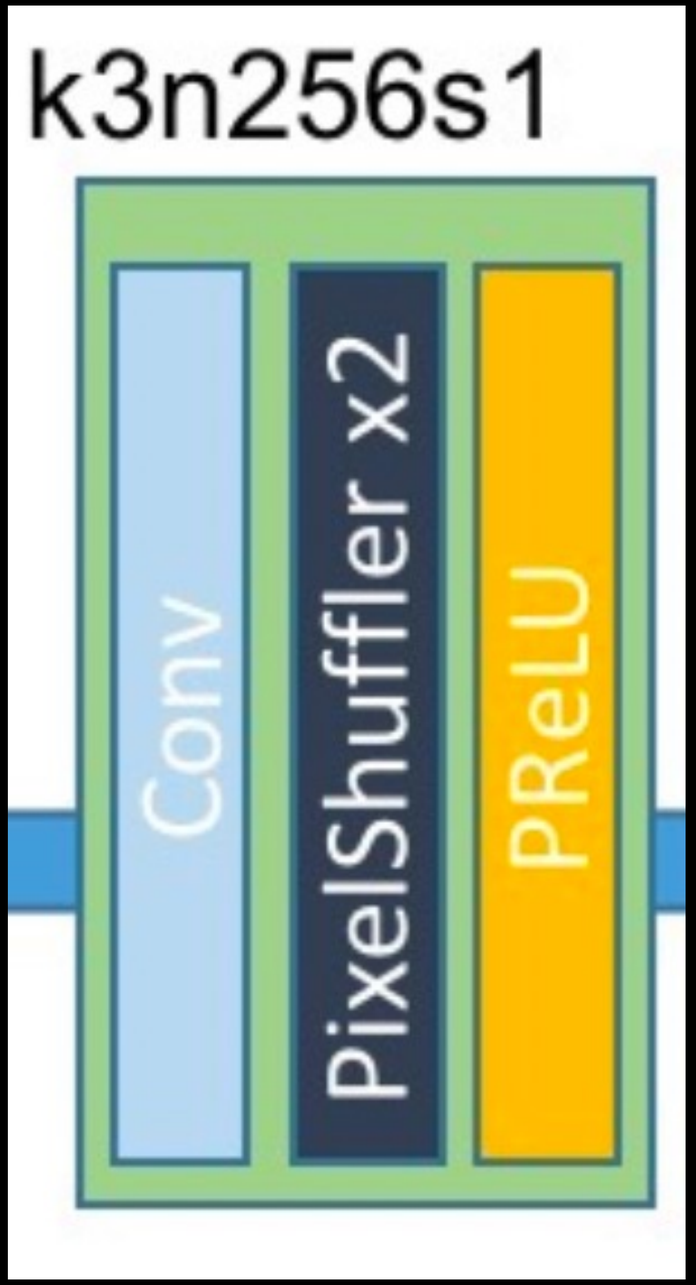


Residual Blocks

- ResNet type architecture
- Each block contains:
    A. Two convolutional layers
    B. Batch normalisation layers after each convolution
    C. PReLU(Parametric ReLU) layer
    D. Skip-Connections
- There are B(hyper-parameter) such blocks



Sub-Pixel Convolutions

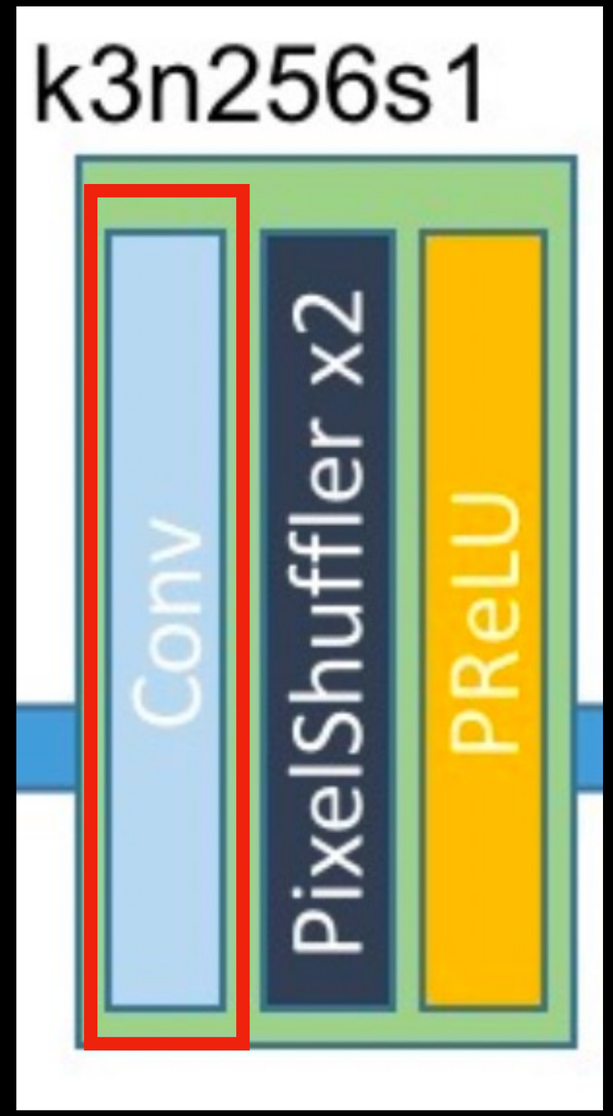


Sub-Pixel Convolutions

- This is used to upscale images
- It can be thought of as a two-step process

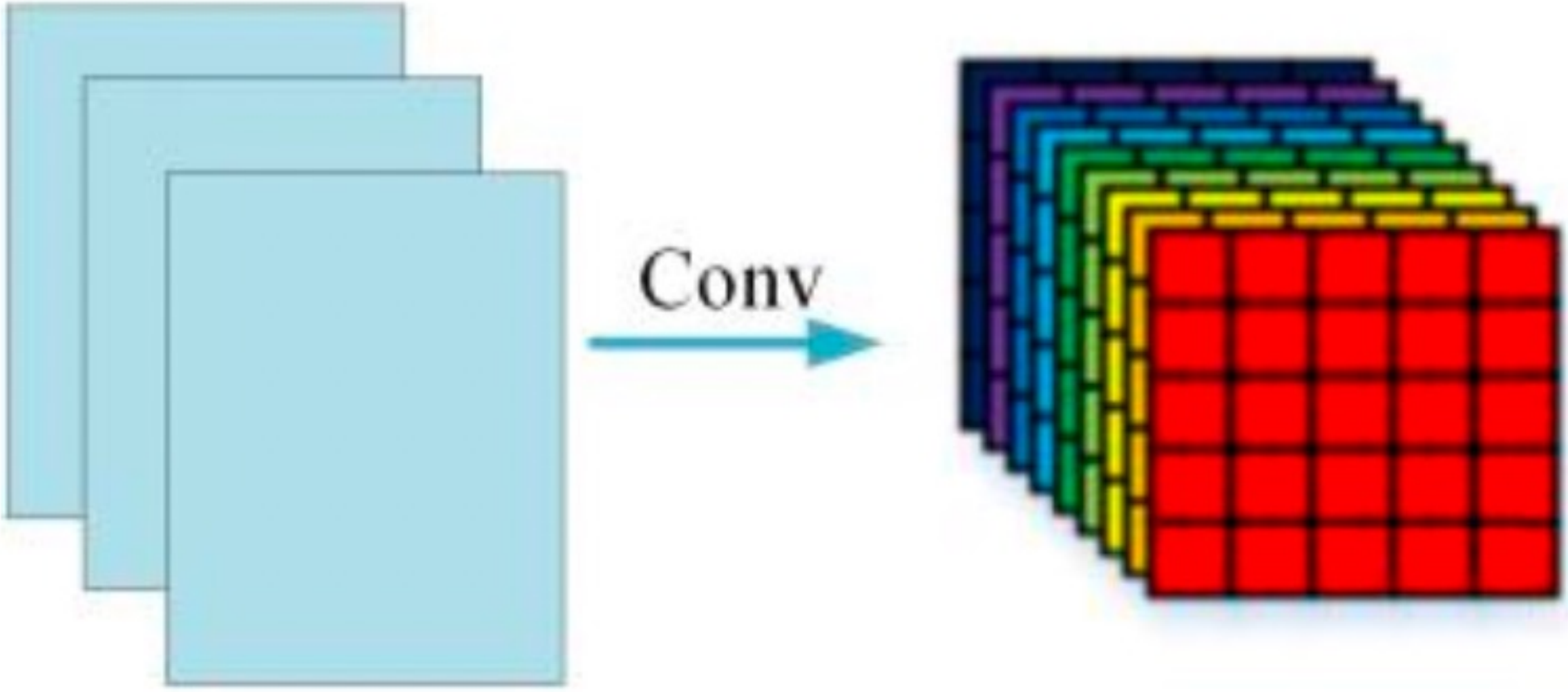
Step 1: A convolution to increase number of channels

$$(C, H, W) \rightarrow (r^2 C, H, W)$$



feature maps

$r \times r$ channels

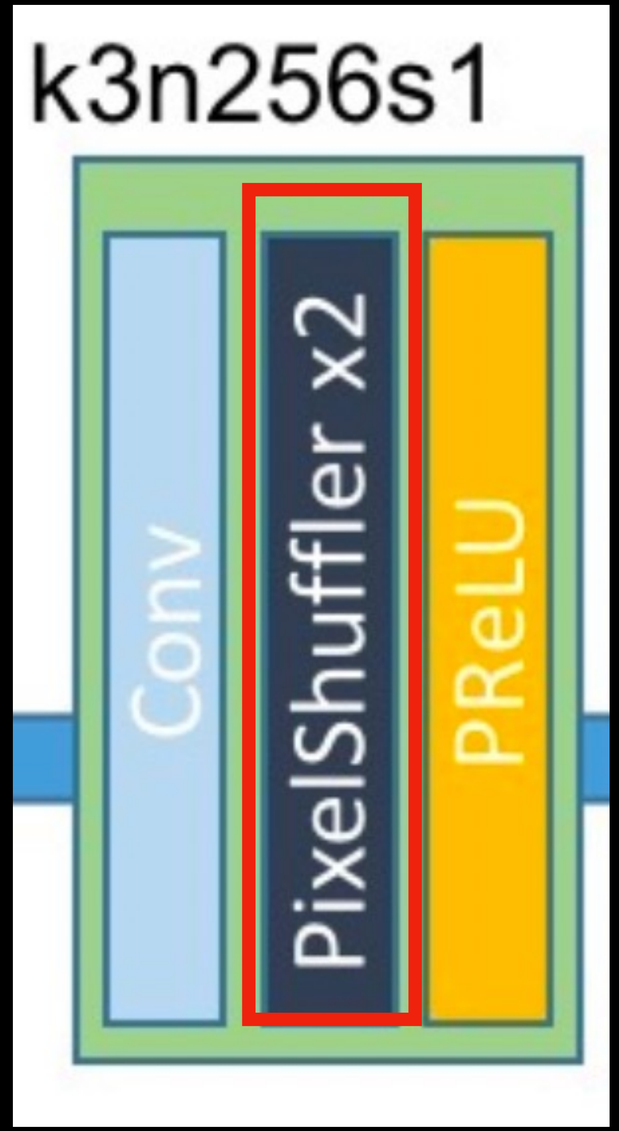


Sub-Pixel Convolutions

Step 2: Pixels from different channels are rearranged to increase resolution

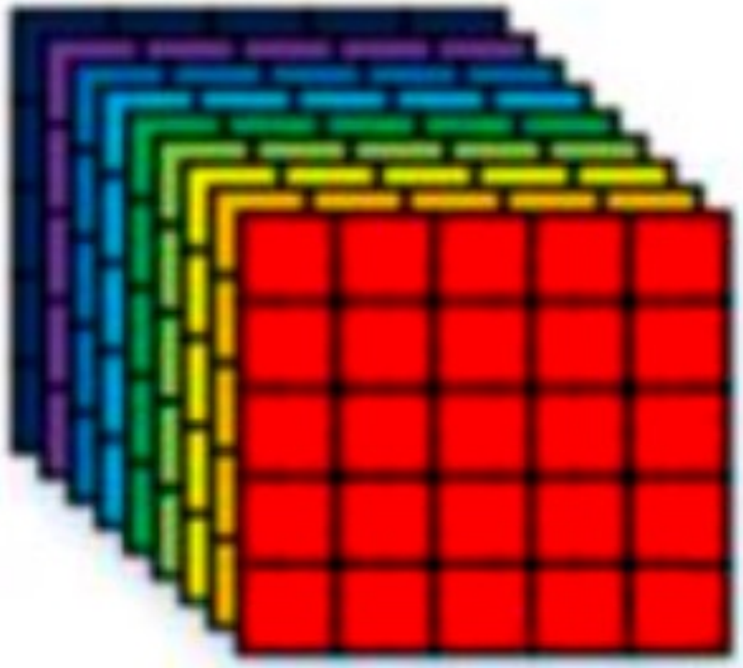
$$(r^2C, H, W) \rightarrow (C, rH, rW)$$

- r is the scaling factor
- These are more commonly called Pixel Shuffle layers.

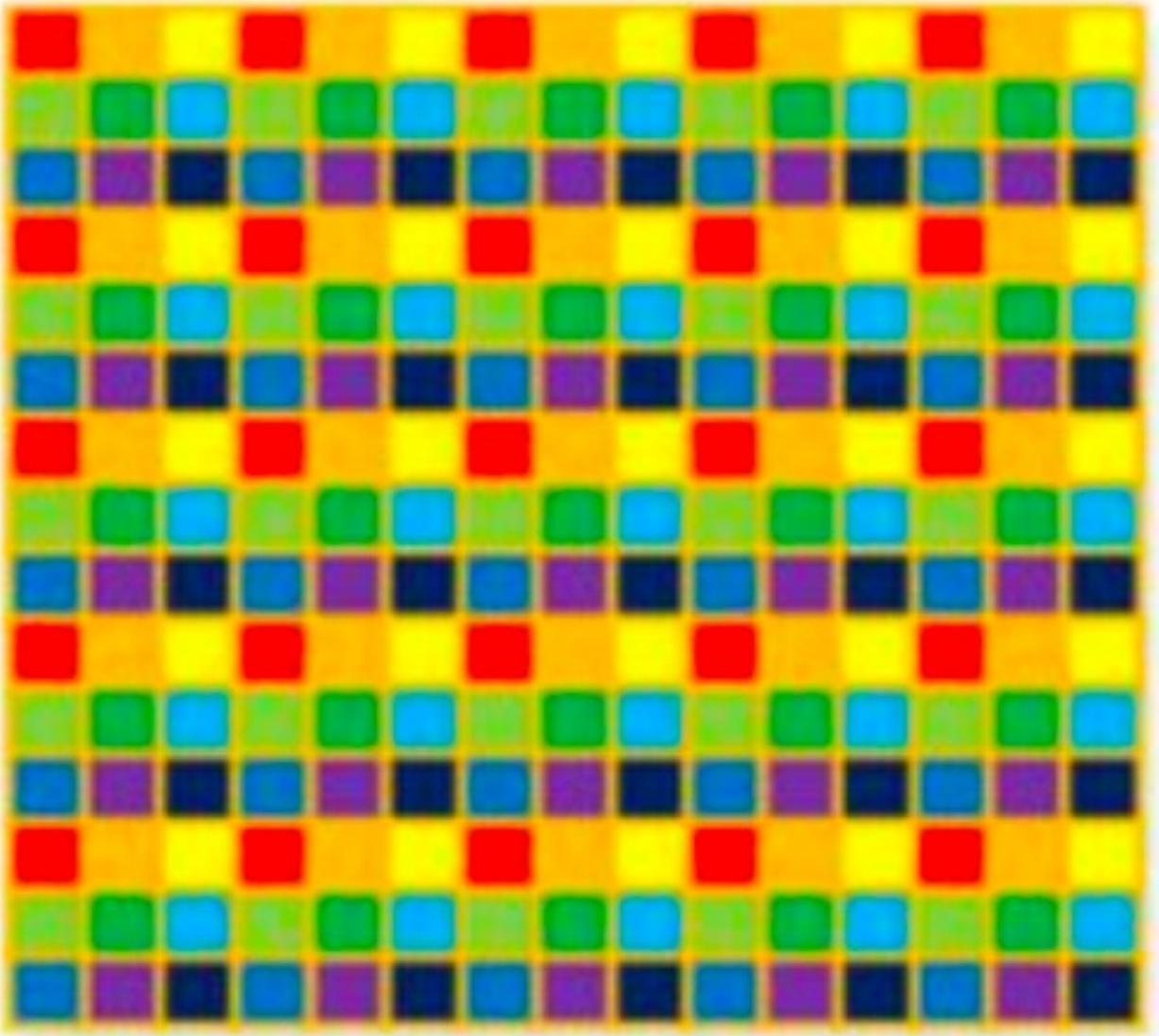


r × r channels

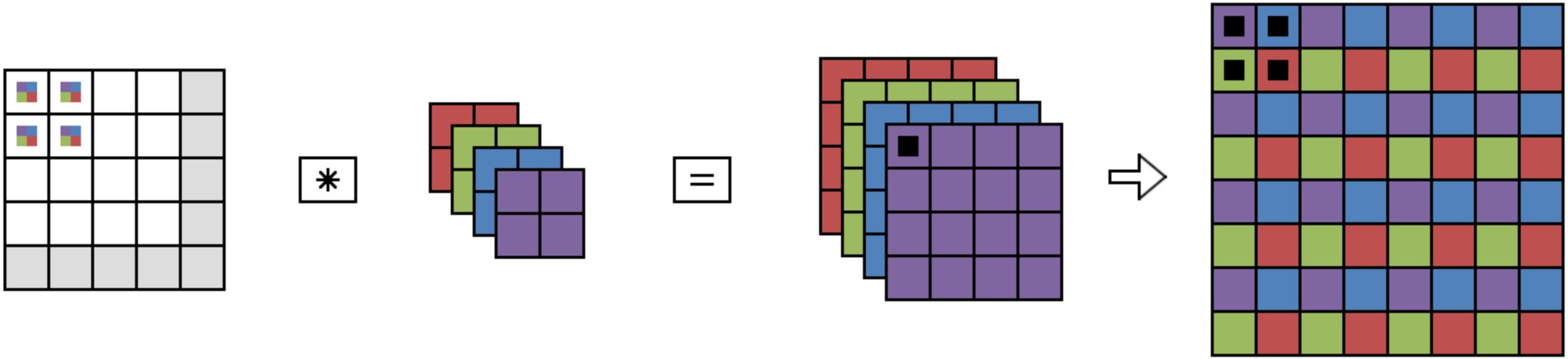
HR Image



rearrange

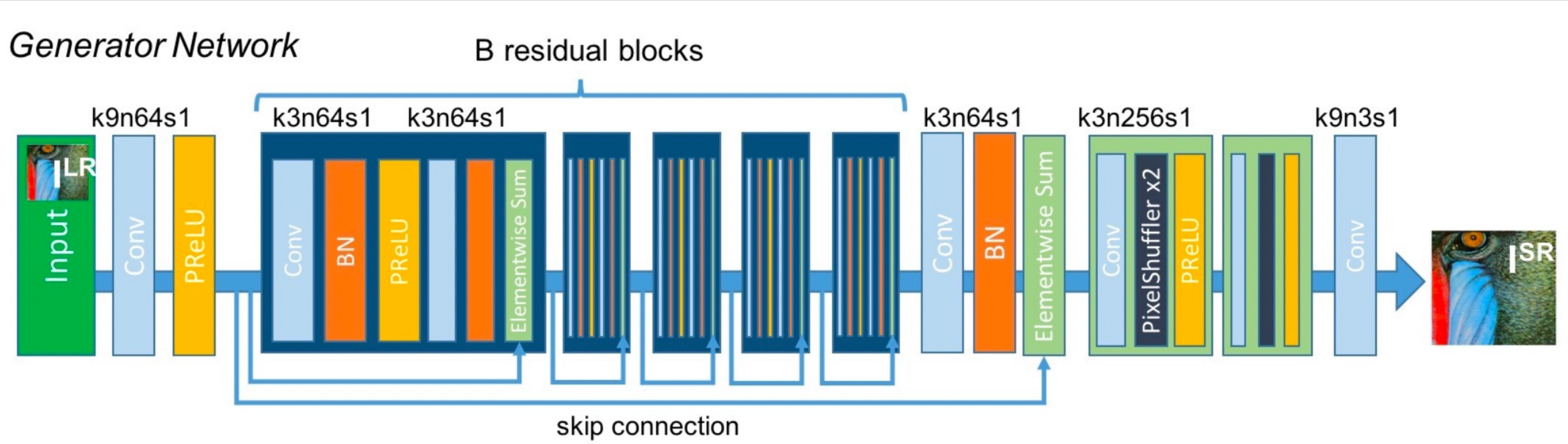


Sub-Pixel Convolutions



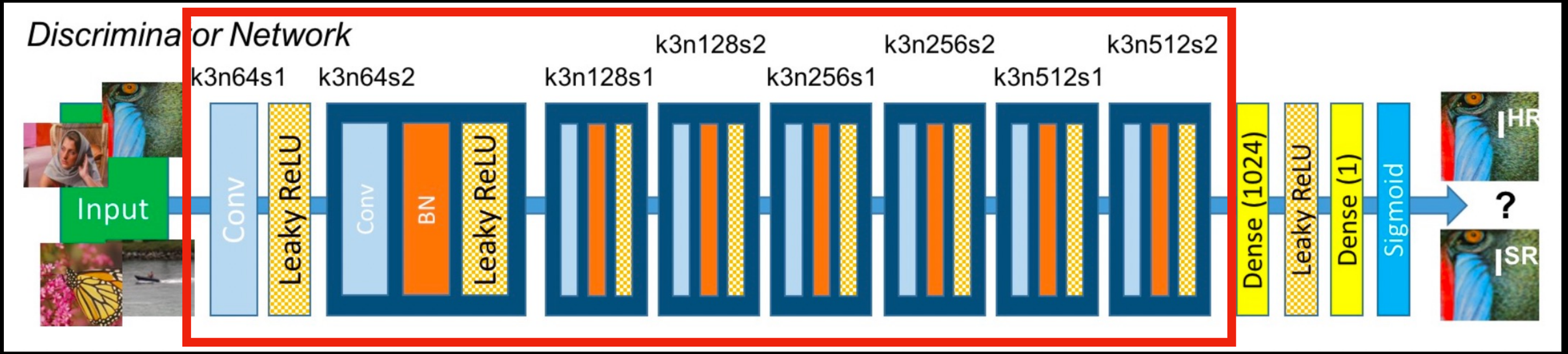
Quick Question

What is the output size of the generator network?
Given the convolutional layers use same padding and the input image has size (3 x 96 x 96)



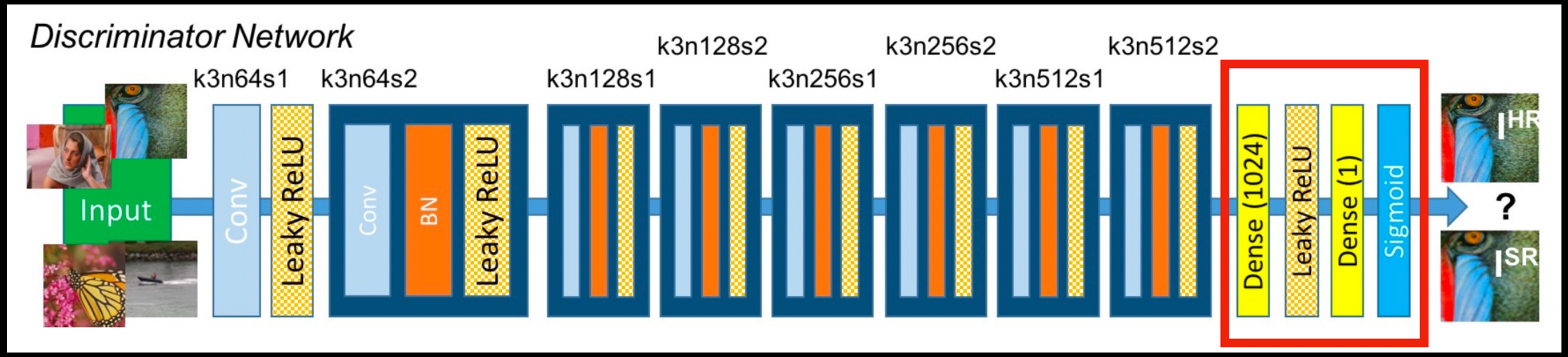
Discriminator Architecture

Discriminator Architecture



- The discriminator tries to distinguish between real and super-resolved images
- Simple classification type architecture
- A set of convolutional layers in which the number of features is increased to 512

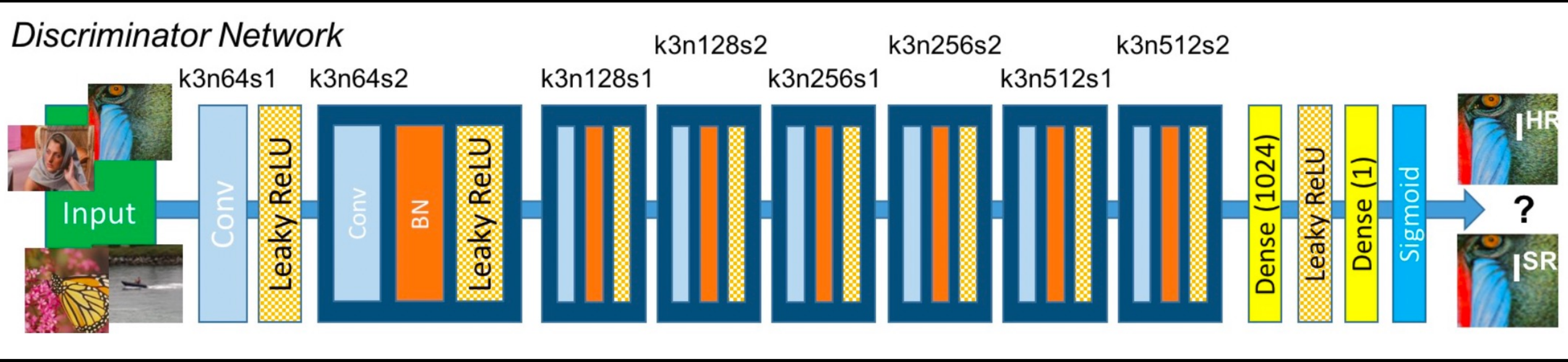
Discriminator Architecture



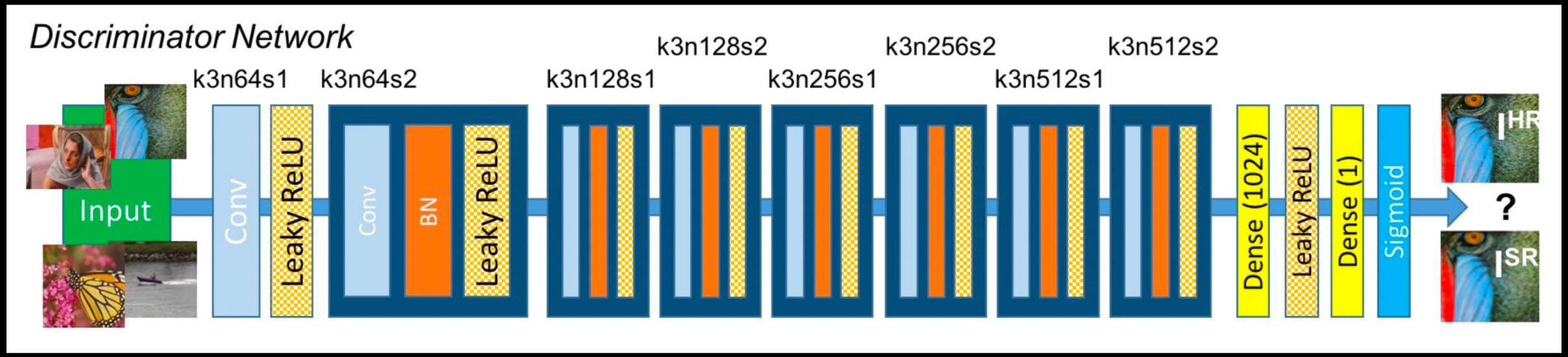
- The output is flattened and passed through fully connected layers
- The linear layers are followed by a sigmoid layer

Quick Question

What does the output of the discriminator represent?
(Think of what the discriminator is trying to achieve)



Discriminator Architecture



- The output from the discriminator is the probability that the image is real and not super-resolved

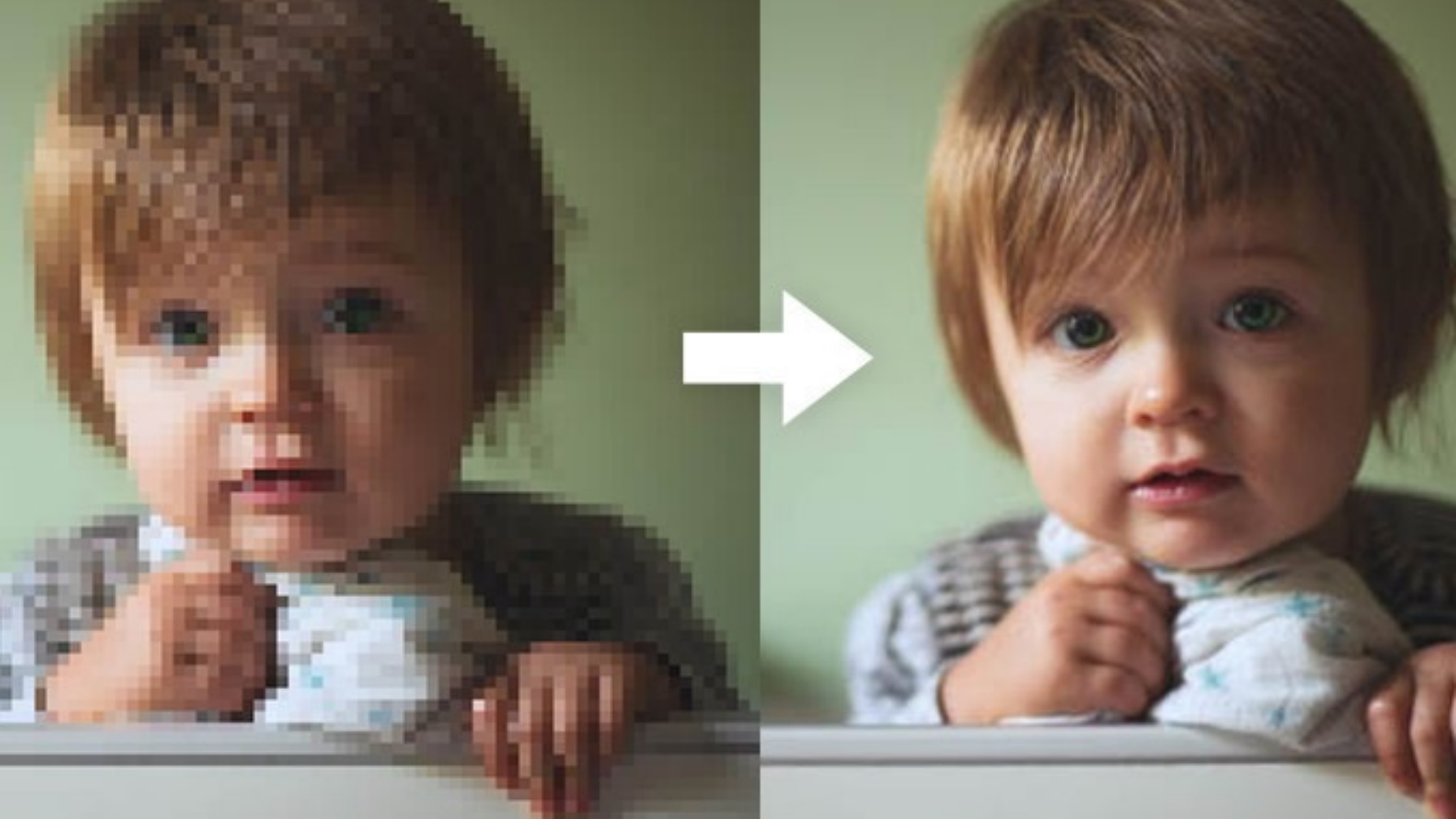
Adversarial Min-Max Problem

Adversarial Min-Max Problem

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

- The expression represents the probability the images are categorised properly, i.e real as real and SR as SR
- For a good discriminator, this must be maximized
- For a good generator, this must be minimized.

How to compare two images?



How to compare two images?

Traditionally two metrics are used to compare two images:

1. Mean Squared Error(MSE)

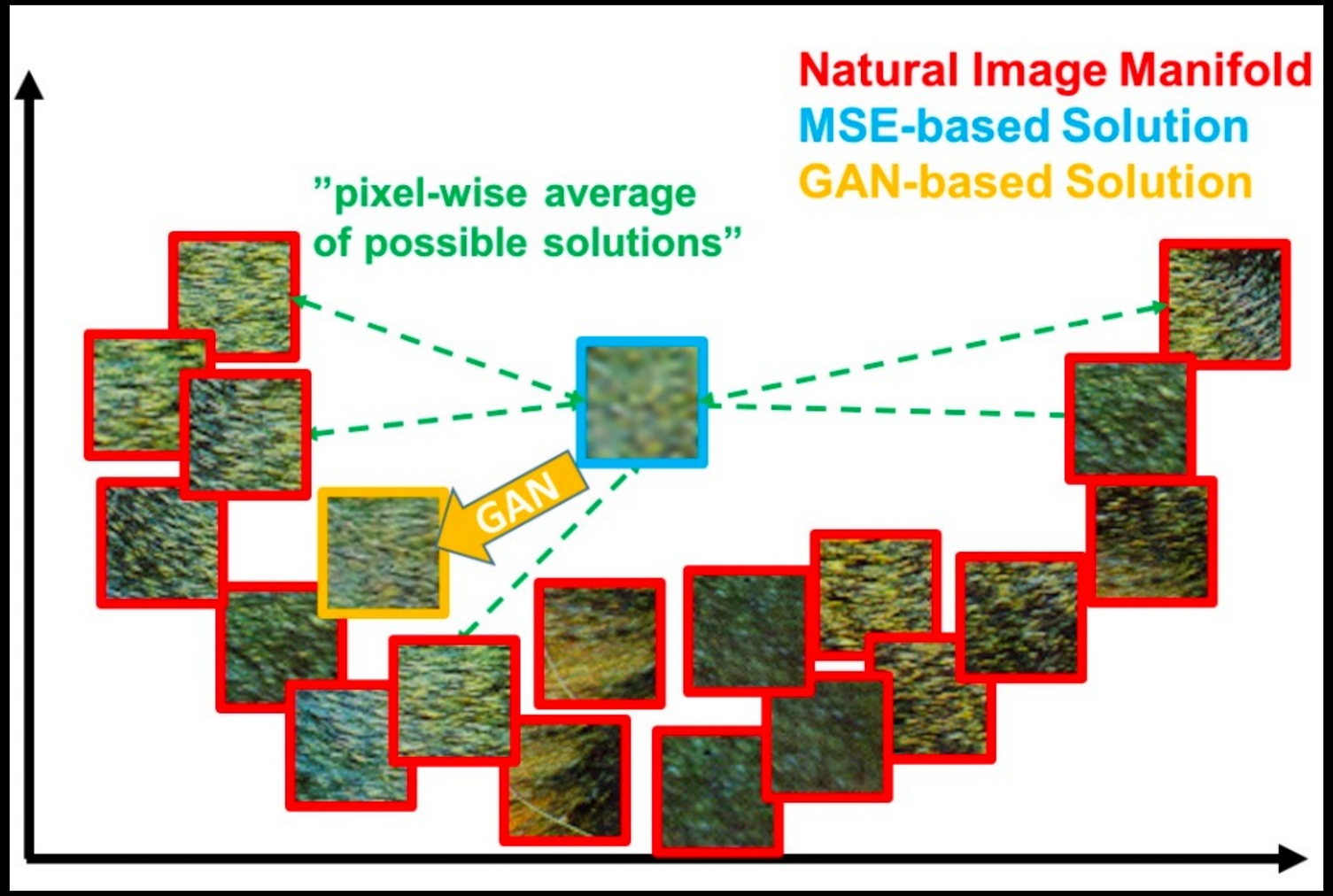
$$l_{MSE}^{SR} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

2. Peak Signal-to-Noise Ratio(PSNR)

$$PSNR = 20\log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right)$$

MAX_f is the maximum signal value in the original image.

How good are these metrics?



- A good SR has low MSE and high PSNR
- However, these metrics deal with the qualities of each individual pixel
- They do not consider visually perceptible attributes like the textures of objects in an image.

So, we introduce another loss function,
Perceptual Loss

Perceptual Loss

- Weighted sum of content loss and adversarial loss

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

- Content loss deals with visually perceptible attributes
- Adversarial loss deals with fooling the discriminator

Content Loss

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- Using MSE for content loss, leads to overly smooth images that are visually unappealing
- So, we define a new loss based on the feature maps of a pre-trained VGG19 network
- It is defined as the euclidean distance between the feature maps of the original image and the super-resolved image
- This is called VGG Loss

Why does VGG Loss work?

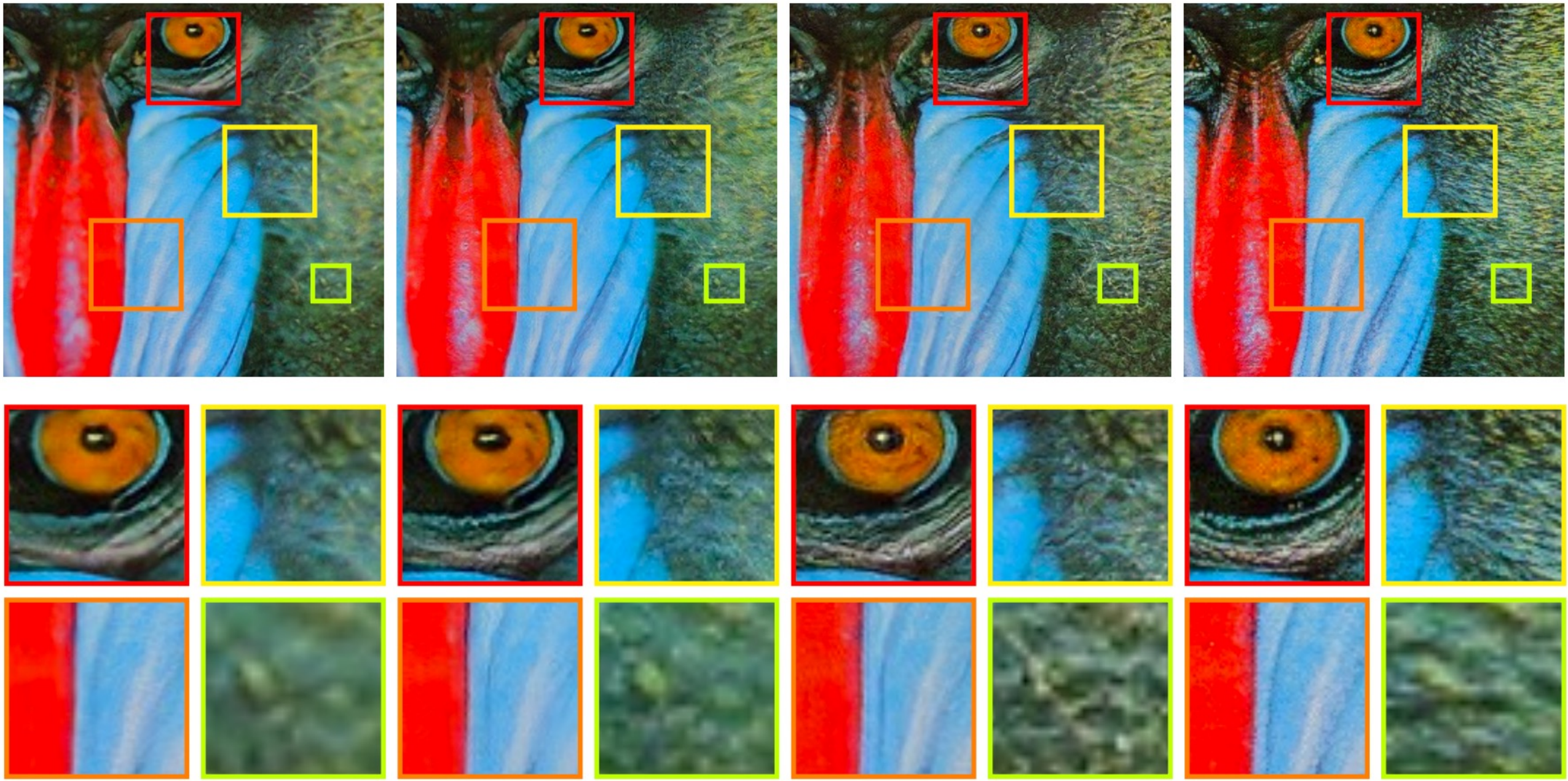
- Any CNN tries to capture the features of an image
- So, if two images are similar it is likely that they have similar feature maps as well
- So, by reducing the distance between the feature maps of the high-res image and the super-resolved image, we can make the super-res image closer to the high-res image

VGG Loss

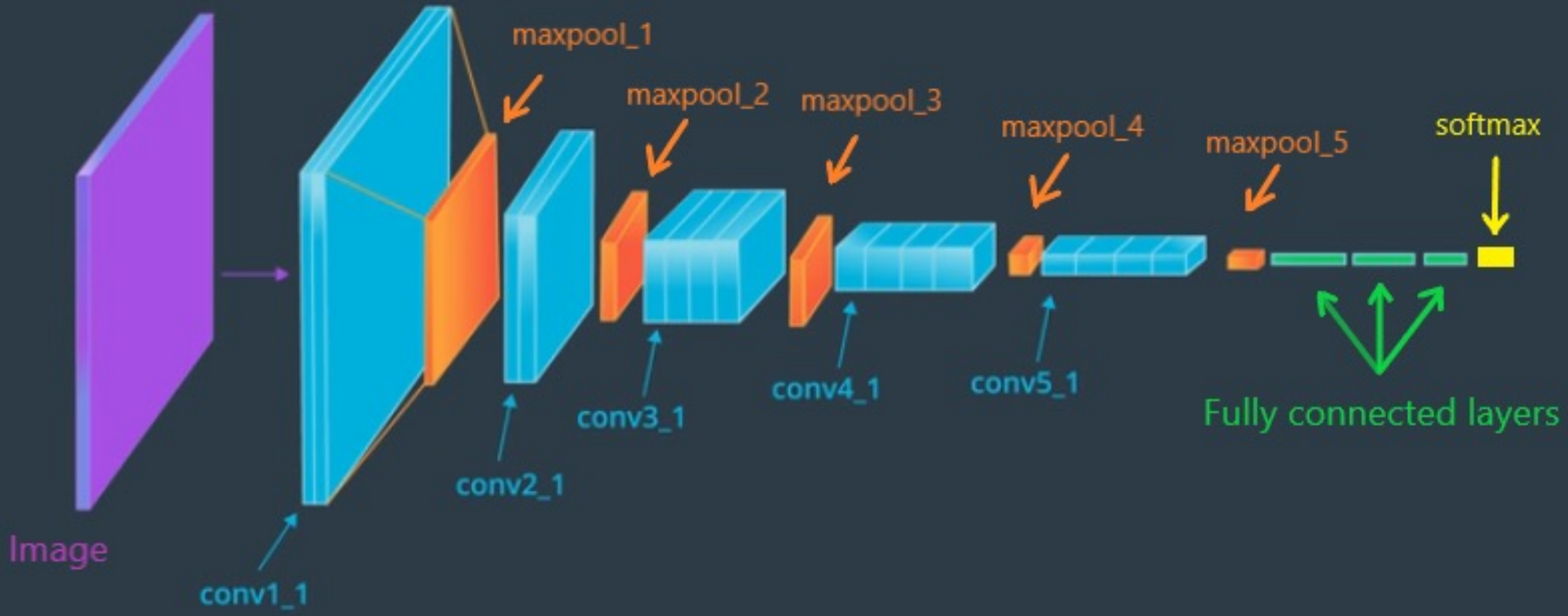
$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

- $\phi_{i.j}$ represents the feature map obtained from the VGG19 network after jth convolutional layer stack and before the i-th max-pooling.
- $W_{i,j}$ and $H_{i,j}$ are the dimensions of the feature map.
- It is observed that higher level features($l^{SR}_{VGG/5.4}$) yield better texture details than lower level features($l^{SR}_{VGG/2.2}$)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					


SRGAN-MSE
SRGAN-VGG22
SRGAN-VGG54
original HR image

VGG - 19 Architecture



Adversarial Loss

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

- In addition to VGG loss, we add the generative loss to the perceptual loss function
- This makes the generator to favour outputs in the original image space in order to fool the discriminator

Training the model

Training the discriminator

- We alternatively train the discriminator and the generator
- The inputs to the discriminator are the high-res images and the outputs of the generator
- We train the discriminator to minimize the adversarial loss

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

Training the generator

- The inputs to the generator are low-res images obtained from the high-res images by adding a gaussian filter and performing a downsampling operation.
- We train the generator to minimize the perceptual loss

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

A comparison between various methods

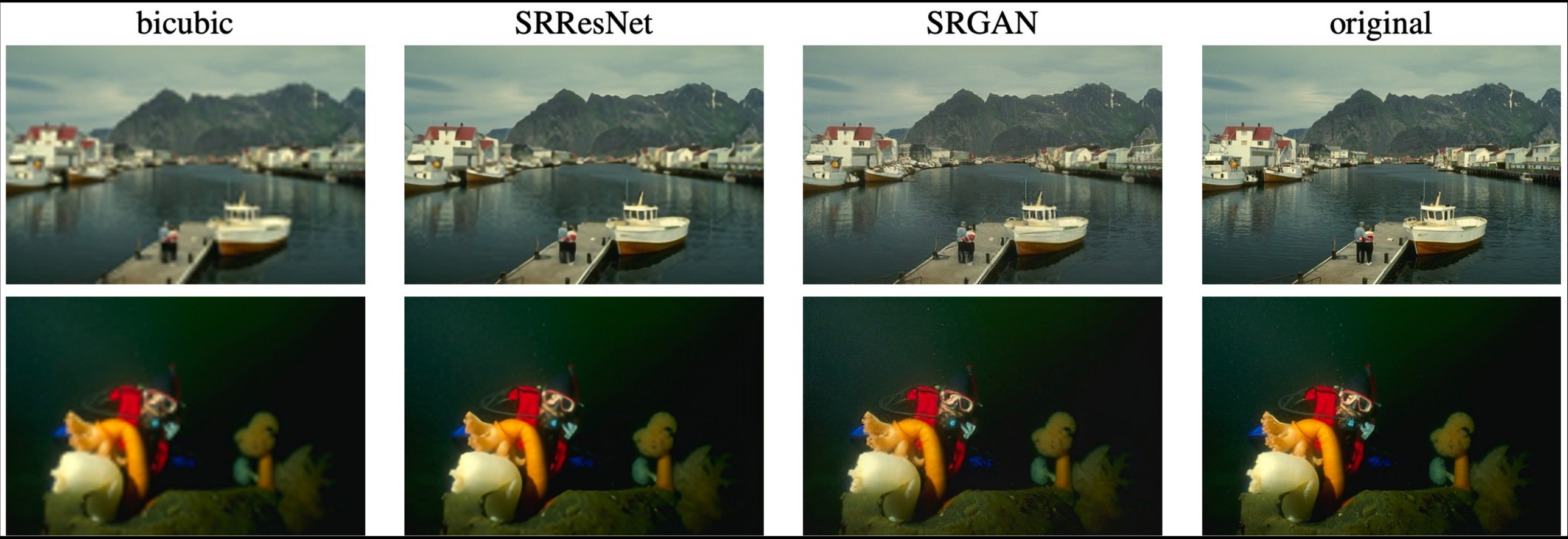
bicubic
(21.59dB/0.6423)

SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

A comparison between various methods



Thank You!