Atharva Joshi
FSRCNN results

hyperparameters:

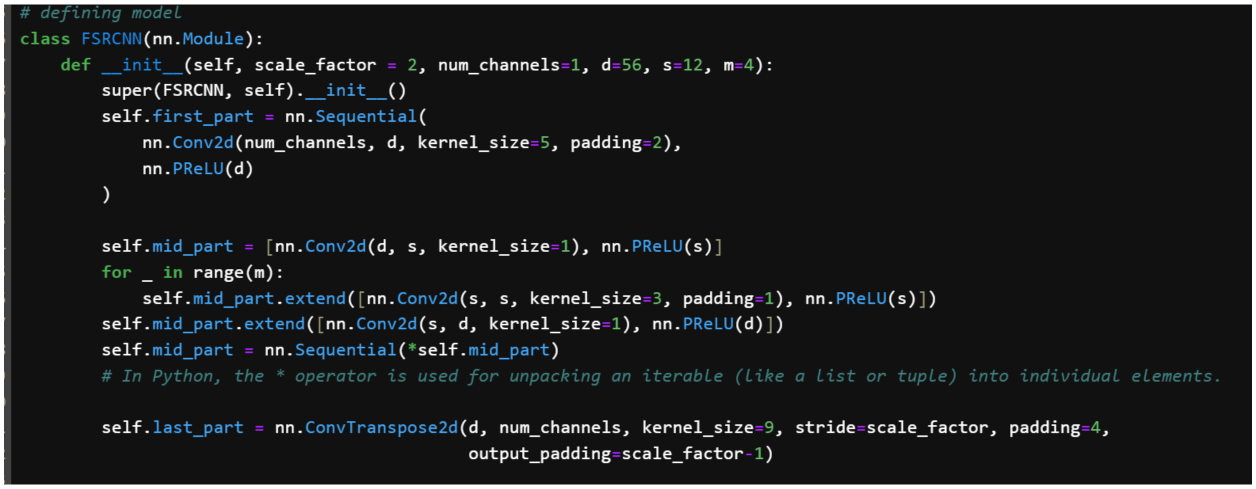
learning rate:
for the first 15 epochs: lr = 1e-3 for first two parts (feature extraction, shrinking, non linear mapping and expansion) and lr*0.1 = 1e-4 for the last part (deconvolution)
for the next 10 epochs: lr = 0.5e-3
for the next 5 epochs: lr = 0.38e-3

loss function:
MSE perpixel loss + perceptual loss with vgg16 pretrained model with equal weightage

batch size = 16
gradient accumulation steps = 3
hence effective batch size = 48

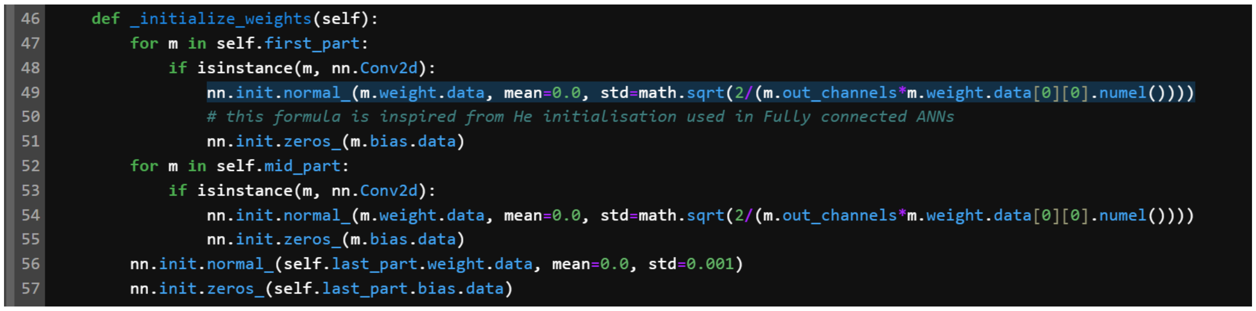
used ycbcr channel separation and trained a y-only model
Optimiser used: Adam

Average PSNR: 24.8 dB

Model architecture:


```python
# defining model
class FSRCNN(nn.Module):
    def __init__(self, scale_factor = 2, num_channels=1, d=56, s=12, m=4):
        super(FSRCNN, self).__init__()
        self.first_part = nn.Sequential(
            nn.Conv2d(num_channels, d, kernel_size=5, padding=2),
            nn.PReLU(d)
        )

        self.mid_part = [nn.Conv2d(d, s, kernel_size=1), nn.PReLU(s)]
        for _ in range(m):
            self.mid_part.extend([nn.Conv2d(s, s, kernel_size=3, padding=1), nn.PReLU(s)])
        self.mid_part.extend([nn.Conv2d(s, d, kernel_size=1), nn.PReLU(d)])
        self.mid_part = nn.Sequential(*self.mid_part)
        # In Python, the * operator is used for unpacking an iterable (like a list or tuple) into individual elements.

        self.last_part = nn.ConvTranspose2d(d, num_channels, kernel_size=9, stride=scale_factor, padding=4,
                                            output_padding=scale_factor-1)
```

Initialisation step:
Kaiming-He initialisation method for all layers except deconvolution layer
for deconvolution weights are intiialised from normal distribution with mean = 0 and std = 0.001



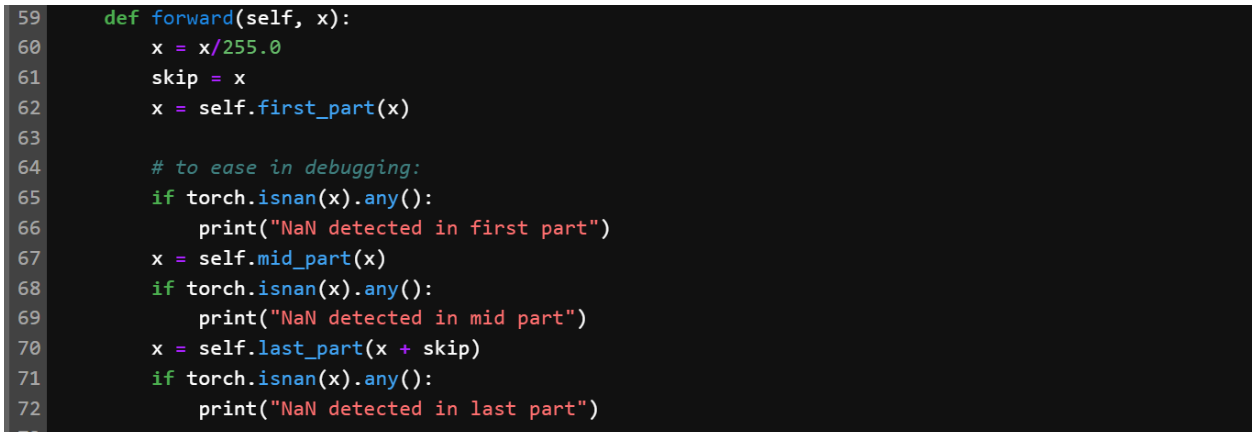
```
    def _initialize_weights(self):
        for m in self.first_part:
            if isinstance(m, nn.Conv2d):
                nn.init.normal_(m.weight.data, mean=0.0, std=math.sqrt(2/(m.out_channels*m.weight.data[0][0].numel())))
                # this formula is inspired from He initialisation used in Fully connected ANNs
                nn.init.zeros_(m.bias.data)
        for m in self.mid_part:
            if isinstance(m, nn.Conv2d):
                nn.init.normal_(m.weight.data, mean=0.0, std=math.sqrt(2/(m.out_channels*m.weight.data[0][0].numel())))
                nn.init.zeros_(m.bias.data)
        nn.init.normal_(self.last_part.weight.data, mean=0.0, std=0.001)
        nn.init.zeros_(self.last_part.bias.data)
```

Forward propagation:

Downscalling performed to scale all pixel values to lie within [0, 1].
This prevents exploding gradients
As a further precaution, the max_norm() function is used to put a limit to the maximum permissible gradient which is set to 2
Further, to debug for exploding gradients, the torch.isnan() method is used

Skip connection was introduced to the deconvolution layer to better learn a mapping close to the identity mapping. This was done because the nature of super-resolution task results in the image being similar in structure to the target image, with addition of new information. This mapping is hence semantically close to identity mapping.

Although theoretically sound, the use of skip connnection resulted in a decrease in performance for the same number of epochs (average PSNR 22.05 dB). A possible reason might be the increased ease of learning the exact identity mapping instead of a modified mapping close to but not exactly equal to the identity mapping.

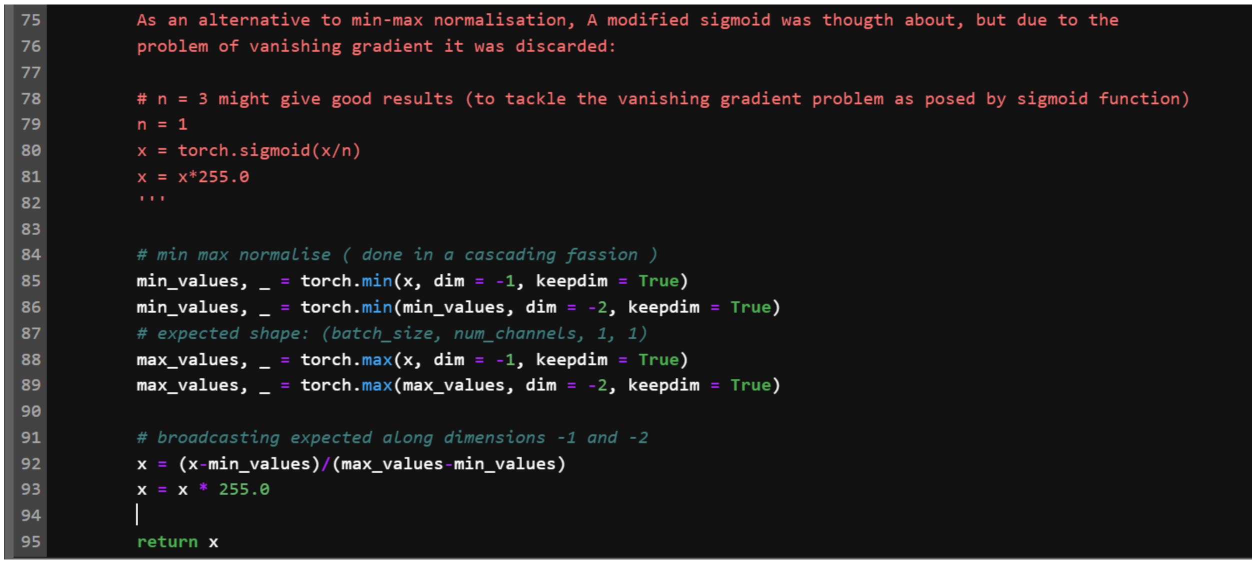


```
    def forward(self, x):
        x = x/255.0
        skip = x
        x = self.first_part(x)

        # to ease in debugging:
        if torch.isnan(x).any():
            print("NaN detected in first part")
        x = self.mid_part(x)
        if torch.isnan(x).any():
            print("NaN detected in mid part")
        x = self.last_part(x + skip)
        if torch.isnan(x).any():
            print("NaN detected in last part")
```

Min max normalisation step:
The min-max normalisation, which scales the output to the range [0, 1], followed by upscaling by 255 ensures that;
1) All pixel values are within the permissible 8-bit range

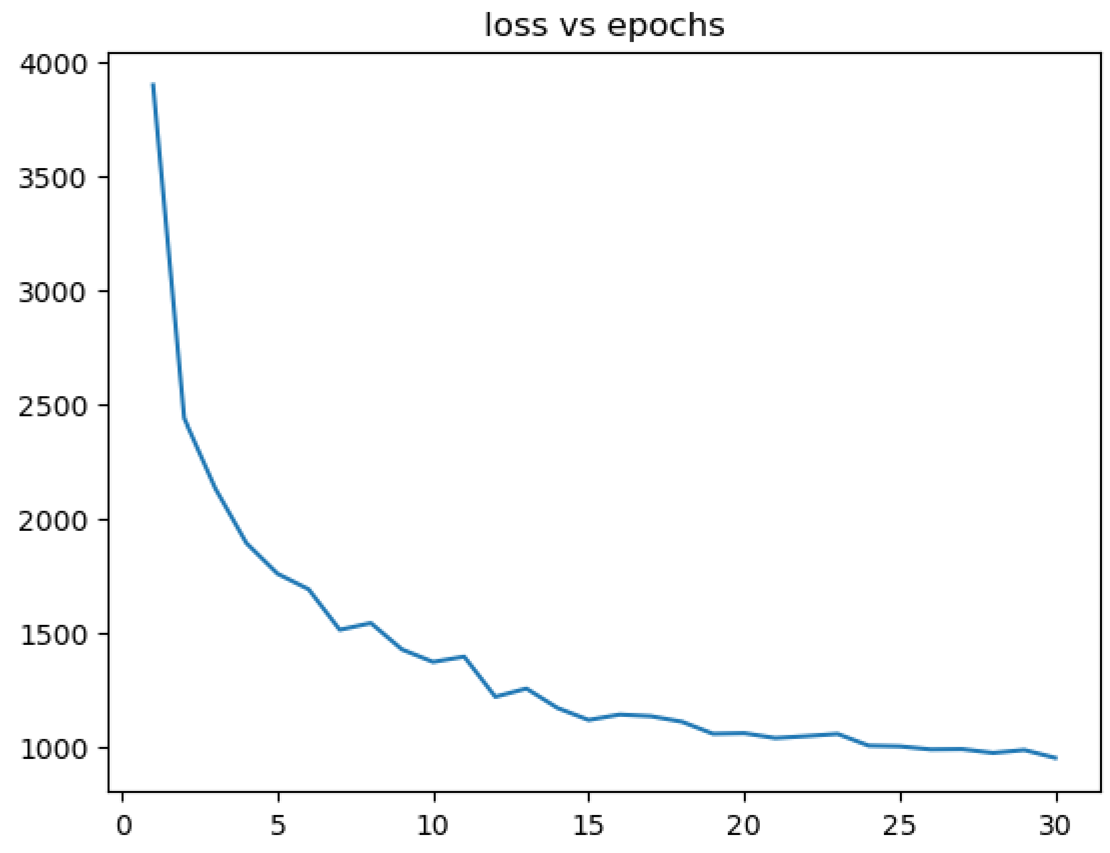
2) The values exceeding 255 in the original model output are scalled linearly to fit within [0, 255] rather than clipping these values. In absence of this function, plain coloured patches were observed

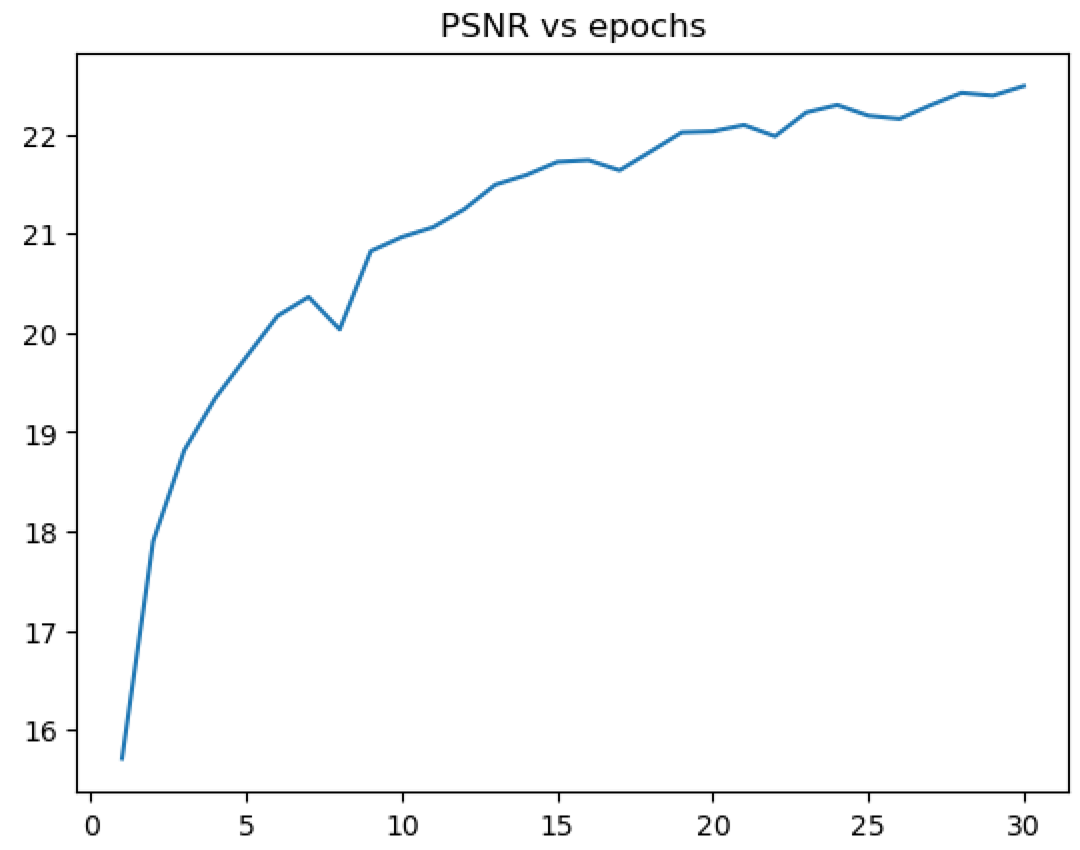
Aliter:
A modified sigmoid function was thought about:
$g(x) = 1/(1 + e^{-x/n})$
with the intuition to 'strech' the graph of sigmoid to tackle vanishing gradient at large values of x. This did not work out. The modification does indeed lead to a relative increase in gradient value at the tail ends, it reduces the absolute value of gradient. Hence the idea was discarded and min-max normalisation was used instead.



```
        As an alternative to min-max normalisation, A modified sigmoid was thougth about, but due to the
        problem of vanishing gradient it was discarded:

        # n = 3 might give good results (to tackle the vanishing gradient problem as posed by sigmoid function)
        n = 1
        x = torch.sigmoid(x/n)
        x = x*255.0
        '''

        # min max normalise ( done in a cascading fassion )
        min_values, _ = torch.min(x, dim = -1, keepdim = True)
        min_values, _ = torch.min(min_values, dim = -2, keepdim = True)
        # expected shape: (batch_size, num_channels, 1, 1)
        max_values, _ = torch.max(x, dim = -1, keepdim = True)
        max_values, _ = torch.max(max_values, dim = -2, keepdim = True)

        # broadcasting expected along dimensions -1 and -2
        x = (x-min_values)/(max_values-min_values)
        x = x * 255.0

        return x
```

Training characteristics:

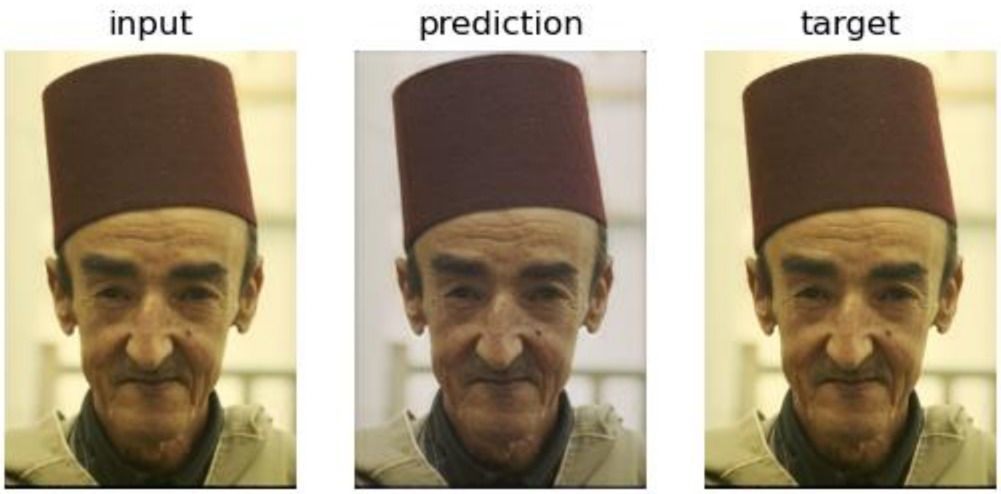




Results:

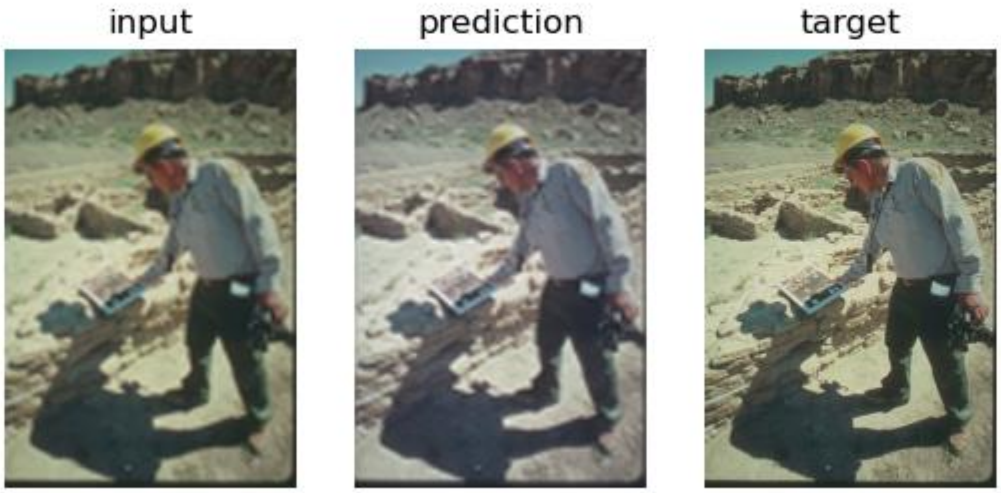
input PSNR: 35.9457

prediction PSNR: 28.5543



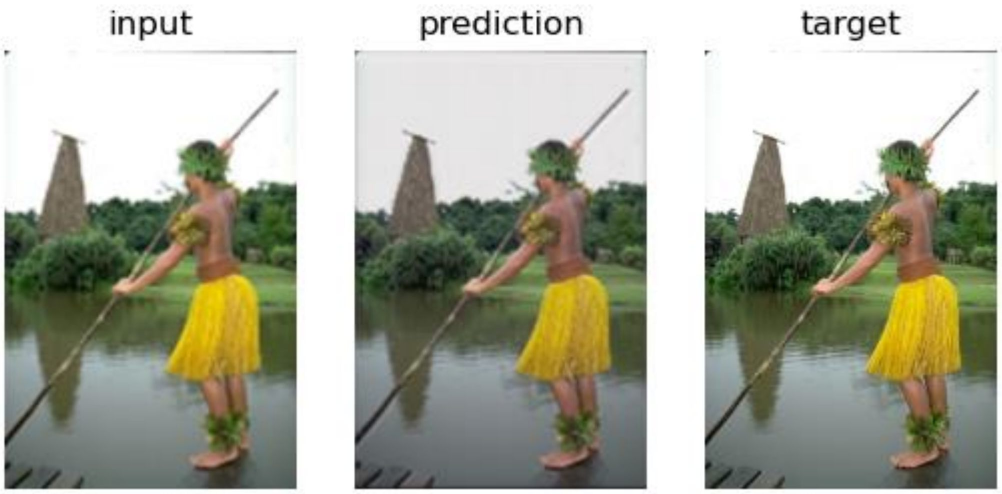
input PSNR: 25.4149

prediction PSNR: 22.4067



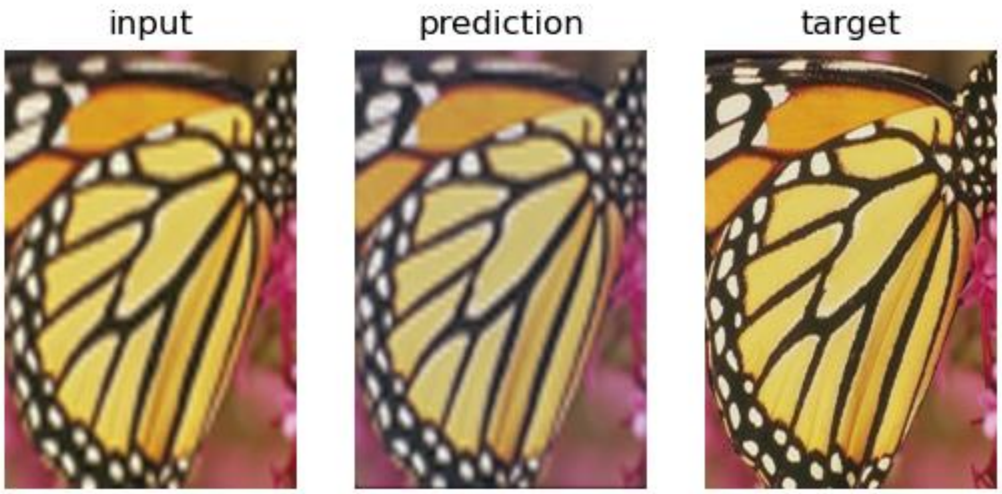
input PSNR: 28.3668

prediction PSNR: 25.7838



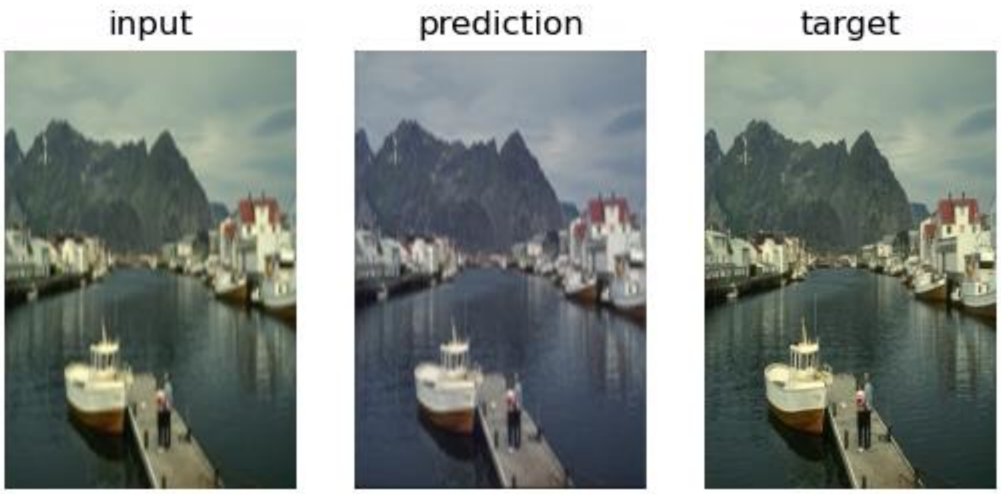
input PSNR: 22.4188

prediction PSNR: 21.9526



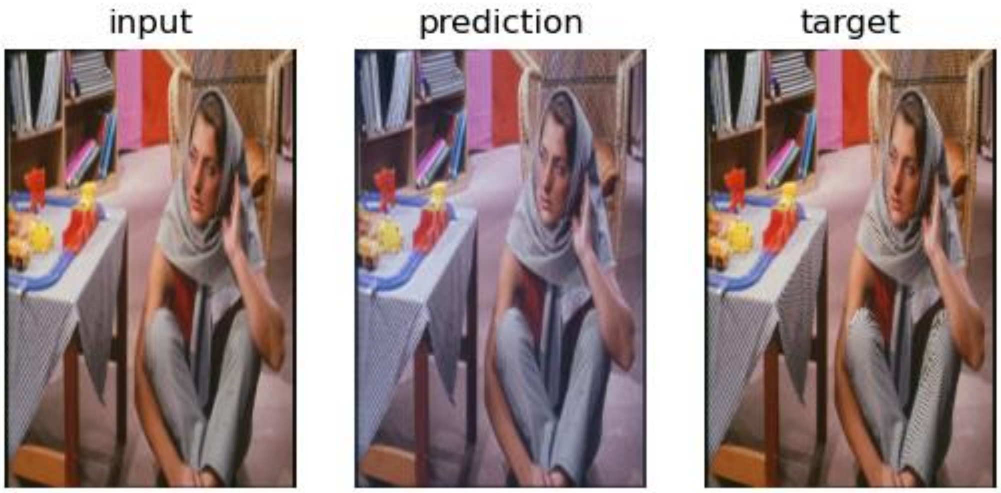
input PSNR: 27.0839

prediction PSNR: 26.3478



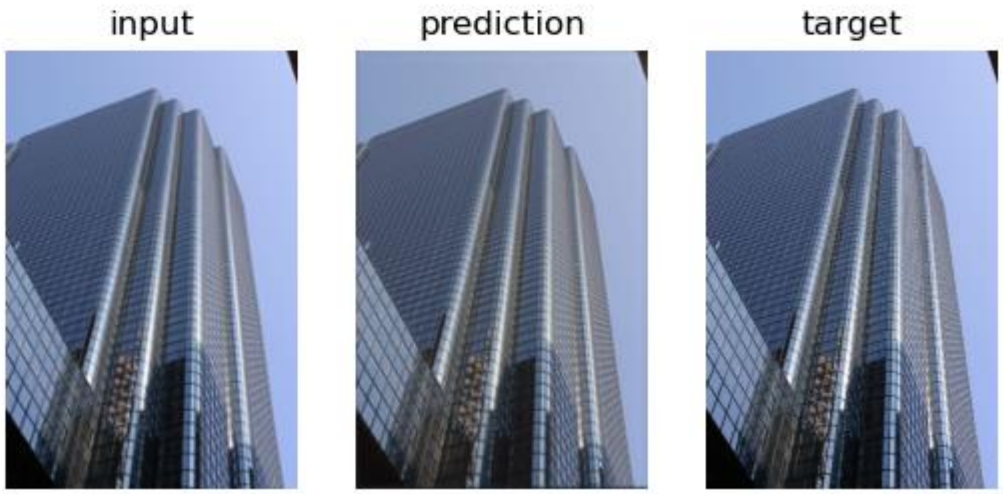
input PSNR: 25.7473

prediction PSNR: 24.3770



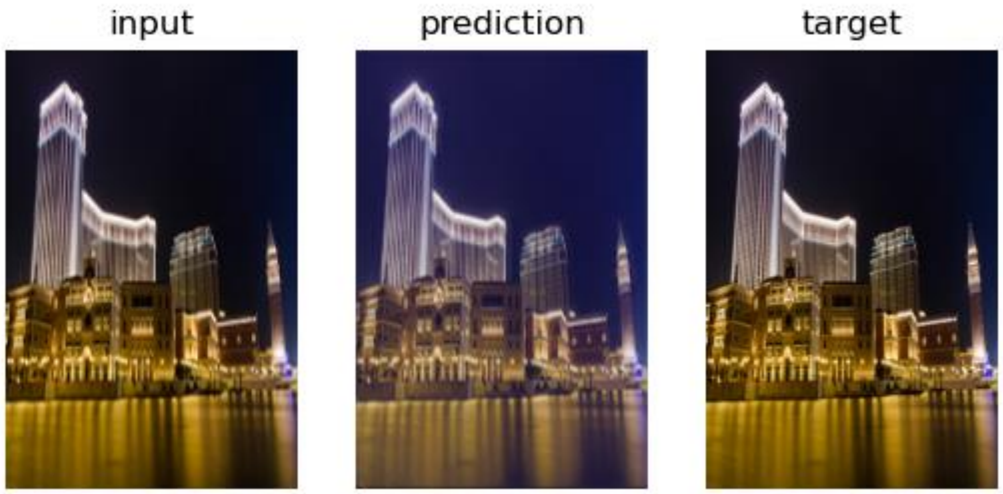
input PSNR: 22.3160

prediction PSNR: 22.0051



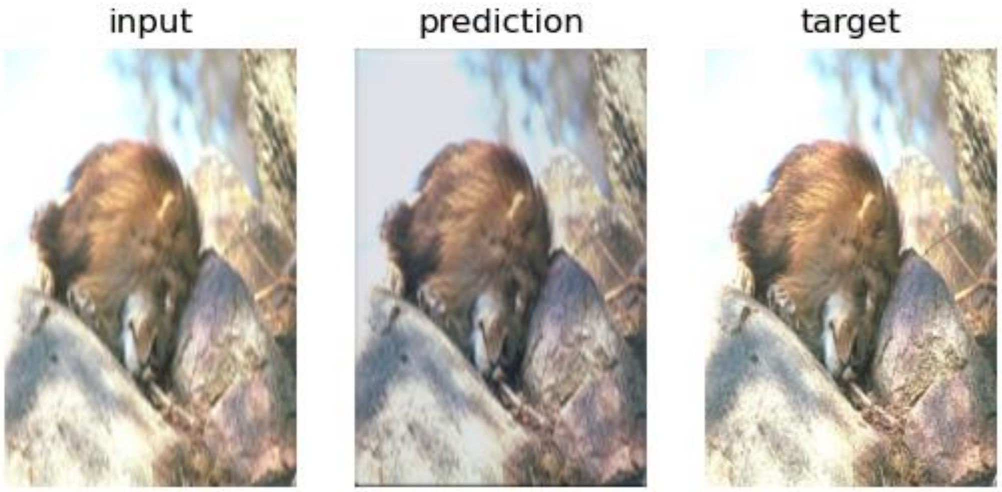
input PSNR: 26.3879

prediction PSNR: 25.5374



input PSNR: 27.1160

prediction PSNR: 22.6743



input PSNR: 19.6747

prediction PSNR: 18.9214