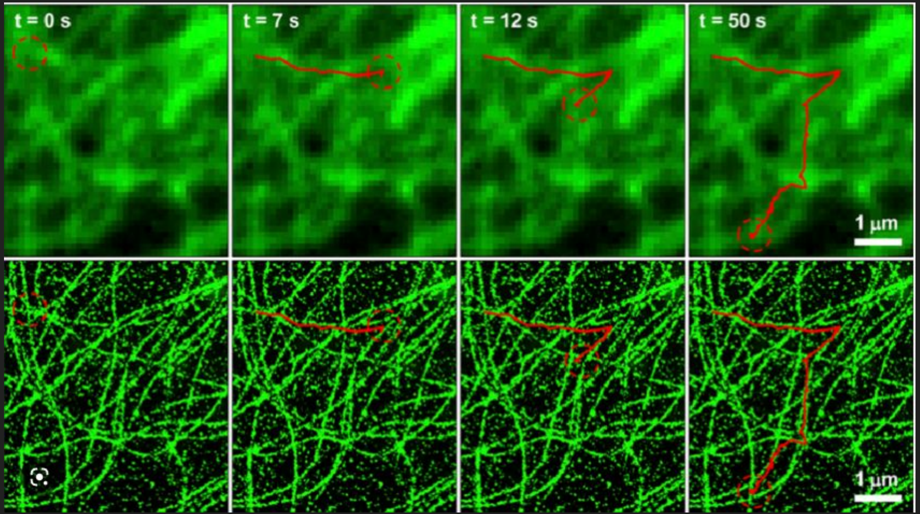
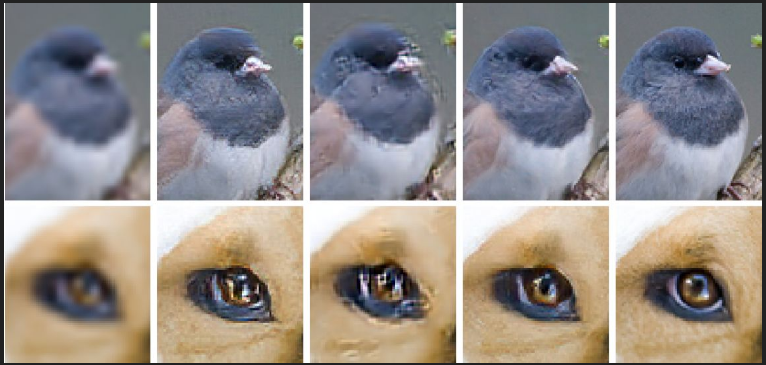
Superresolution with GAN

Emre İnceoğlu
Oğuzhan Atakan

Problem Statement and Motivation

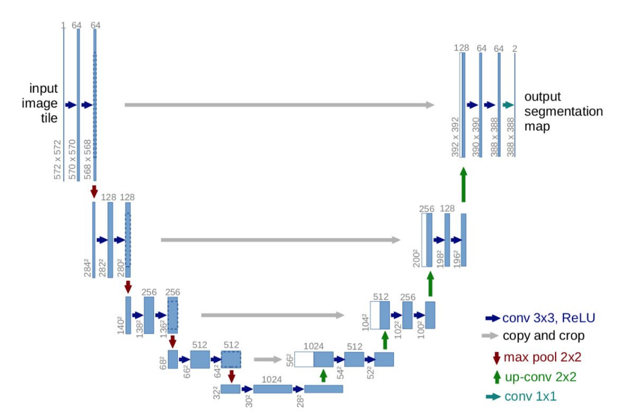
1) It is important to upsample images with min loss in quality - has many applications.
2) Better results than conventional upscaling methods.

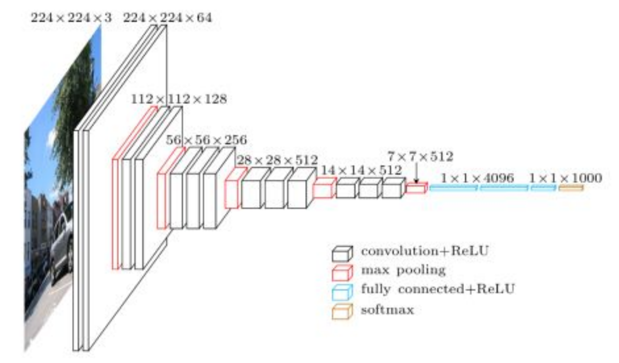




Related Works

1) EnhanceNet: Proposes a method that uses deeply-recursive conv networks that receives a significant boost in upsampling high resolution images.
2) UNet: One of the earlier works for image segmentation. Uses encoding and decoding layers with skip connections.
3) VGGNet: 19 layered convolutional neural network used for image classification.



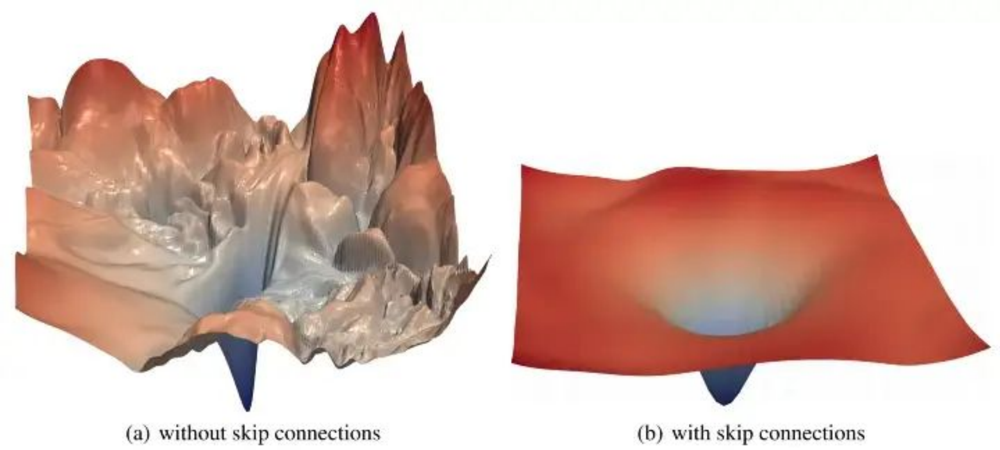


General Methodology

1) We used a generator that takes downsampled image as input and generates the enhanced image. Discriminator is trained to distinguish between fake and real images to achieve better results.
2) Tried Conv-RNN to and Densely Connected Conv Nets to get better results. Later shifted from this approach.
3) Used the VGG network in the calculate loss, which improved our results.
4) Also used TVLoss in our loss function.
5) Experimented with color saturation and ratios in the loss function.

Methodology

1) Implemented residual nets and skip connections, similar to UNet, which improved our performance.
2) Used DIV2K dataset to train our models.



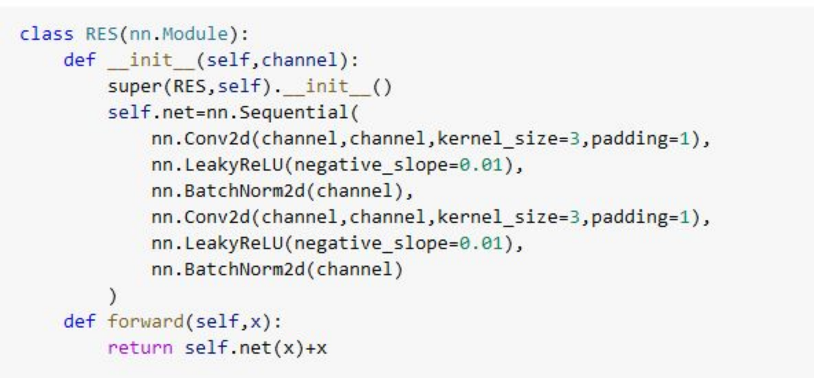
(a) without skip connections

(b) with skip connections

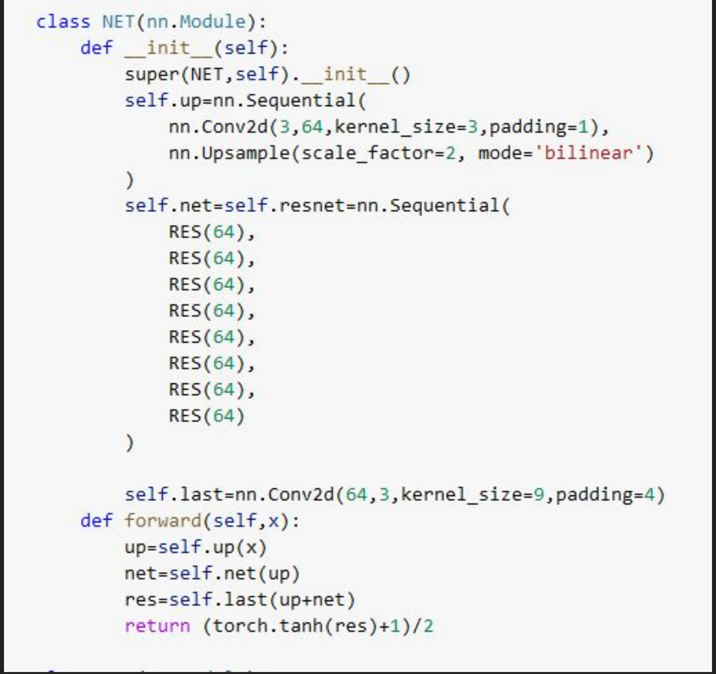
Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Methodology

- We used a small convolutional neural network with skip connections to implement our generator.



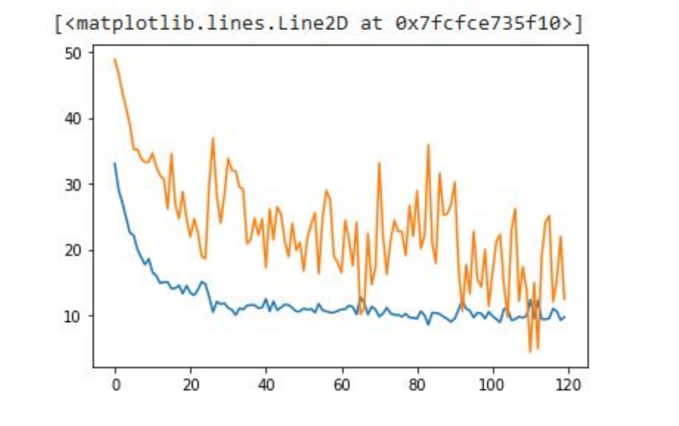
```
class RES(nn.Module):
    def __init__(self,channel):
        super(RES,self).__init__()
        self.net=nn.Sequential(
            nn.Conv2d(channel,channel,kernel_size=3,padding=1),
            nn.LeakyReLU(negative_slope=0.01),
            nn.BatchNorm2d(channel),
            nn.Conv2d(channel,channel,kernel_size=3,padding=1),
            nn.LeakyReLU(negative_slope=0.01),
            nn.BatchNorm2d(channel)
        )
    def forward(self,x):
        return self.net(x)+x
```



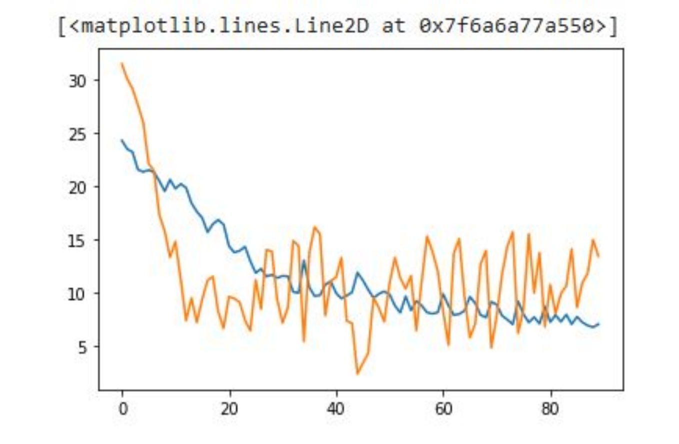
```
class NET(nn.Module):
    def __init__(self):
        super(NET,self).__init__()
        self.up=nn.Sequential(
            nn.Conv2d(3,64,kernel_size=3,padding=1),
            nn.Upsample(scale_factor=2, mode='bilinear')
        )
        self.net=self.resnet=nn.Sequential(
            RES(64),
            RES(64),
            RES(64),
            RES(64),
            RES(64),
            RES(64),
            RES(64),
            RES(64)
        )

        self.last=nn.Conv2d(64,3,kernel_size=9,padding=4)
    def forward(self,x):
        up=self.up(x)
        net=self.net(up)
        res=self.last(up+net)
        return (torch.tanh(res)+1)/2
```

Training

- Used SGD and CyclicLR scheduler in our training.
- Used TVLoss, Custom VGG Losses, Custom color ratio loss and MSE Loss for calculating loss.
- We ran models for 120 epochs, which took less than 3 hours.



```
[<matplotlib.lines.Line2D at 0x7fcfce735f10>]
```



```
[<matplotlib.lines.Line2D at 0x7f6a6a77a550>]
```

Results