Enhancing Low Resolution Faces for Identification Using MPSRGAN

Palos Peter
Budapest University of Technology and Economics

Hevizi Marton
Budapest University of Technology and Economics

Abstract

Upscaling images using traditional methods are not ideal, because of no added detail. They come with pixelation even at low upscaling factors. Convolutional neural networks can be used to analyze low resolution images of faces and generate possible high resolution images for identification purposes. We created an algorithm based on SRGAN and specialized our model to work on faces.

1 Introduction

Enhancement of image resolution comes with higher number of pixels but no added detail. Traditional image upscaling is done by an interpolation algorithm, for example nearest-neighbor or bicubic [5] interpolation. They try to guess the values of new pixels based on the originals, but even at smaller upscaling factors they smooth the image and reduce contrast. To solve these problems, and achieve higher scaling factors we use deep convolutional neural networks [6], which can analyze the picture and add the missing information. We used a generative adversarial network [2] and VGG [8] in order to train the network. Our main goal was to upscale faces for identification, so staying true to the original was important. We didn't want our generator to add features to the created image that weren't there in the originally, for example adding glasses or birthmarks.

2 Related work

Single image super resolution algorithms have evolved a lot in the past. The first methods were different interpolation techniques, which were fast, but not accurate. Training based methods used low and high resolution image pairs to get better results. A big problem with these algorithms were bad edge artifacts on upscaled images.

Many solutions were made to try and solve these problems and convolutional neural network worked well for super resolution. The model was later extended by adding more layers, and to counteract the longer training, batch normalization [4] and residual blocks [3] were added.

The early models were using mean squared error as a loss function, but this averaged the result and created smooth pictures. Generative adversarial networks were used to solve this problem, where a discriminator evaluates the results of the generator, to create more realistic images.

Mean squared error was calculated between the generated high resolution image and the original high resolution image, but extracting features with VGG from both and comparing those features yielded better results.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

3 Architecture

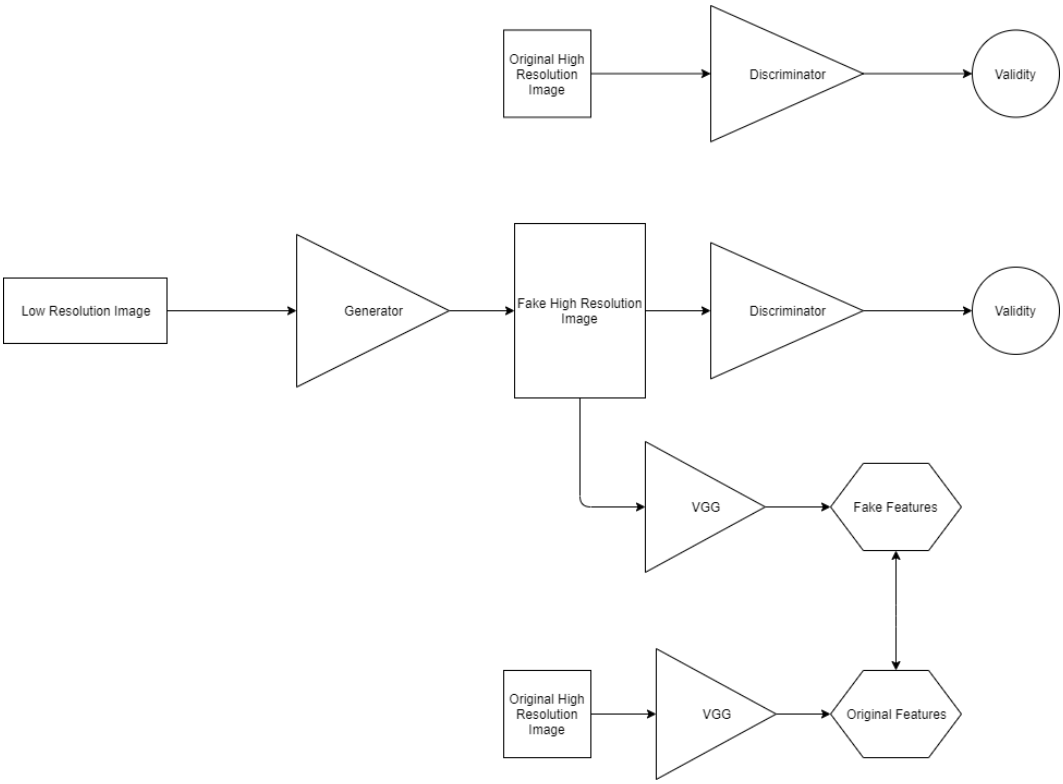

Figure 1: Every element of our model

The basic structure of our network is generative adversarial network, and in our case it consists of three parts. The first is the generator, which creates the high resolution images from low resolution images. Second is the discriminator, it takes the generator's output, the high resolution image, and evaluates it, returning the probability of it coming from the training set. Finally we use VGG, an image classifier network, but only for extracting the features of our high resolution pictures for comparison. Our model is based on SRGAN, they contributed to super resolution a new perceptual loss function which combines the discriminator loss with a mean squared error based content loss on the features extracted by VGG. The input is 64x64 pixels and with and upscaling factor of 4 teh output is 256x2516 pixels.

3.1 GAN

GAN was invented by Ian Goodfellow and Co. in 2014. In the adversarial process two networks are trained, the generator that creates data with a distribution learned in the training process, and the discriminator that tries to evaluate if the sample is from the training data or the generator. The goal of the generator is for the discriminator to make a mistake and wrongly label it's output. With training the generator will be able recreate the training data distribution and the discriminator will have a success rate of 50 percent. When these model are multilayer perceptrons, the system can be trained with backpropagation.

3.2 VGG

VGG is a network using very deep convolutional networks for large-scale image recognition. It uses 16-19 weight layers and very small 3x3 convolutional filters. They showed the positive effects of increasing network depth on classification accuracy. VGG also generalizes well to other data sets, which makes it easy to include in other models.

We are only using it for feature extraction, and not classification and we are not training it any further.

3.3 Generator

Multiple residual blocks create the base of our generator, these are the same in structure. A residual block consists of two convolutional layer with a kernel size of 3x3 and a filter of 64. After every block we use a batch-normalization layer and ReLU activation. With the skip connection in the residual blocks we can counteract the effects of many layers on training time. these help avoiding the problem of vanishing gradient and speed up the training for the inner convolutional layers. The layout was created by Sam Gross and Michael Wilber.

We used two deconvolutional blocks which contain UpSampling2D layers for the actual upscaling of the image.

3.4 Discriminator

We evaluate whether the image is real or generated by the generator with the discriminator network. The architecture was created by Alec Radford [7] and Luke Metz. It uses convolutional layers with LeakyReLU activation. Each successive layer increases the filter size. A final Dense layer generates our output, the probability of our input being real.

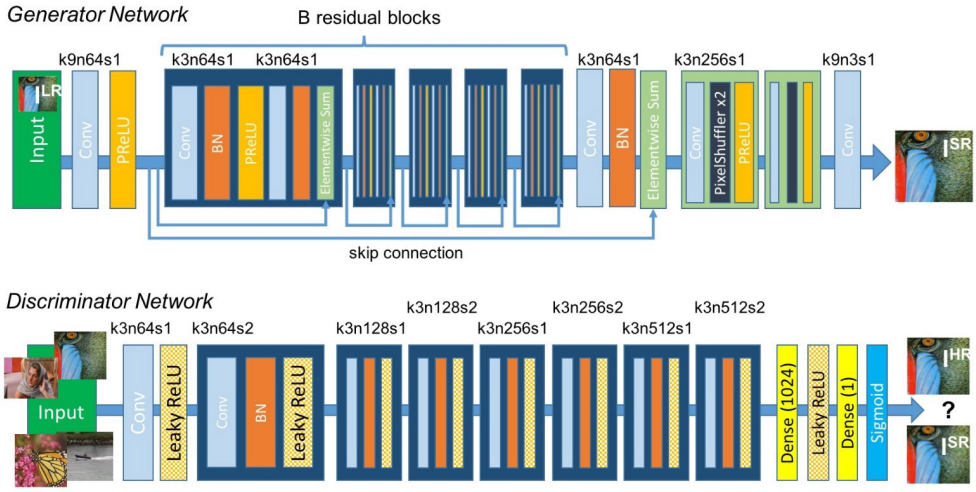

Figure 2: The generator and discriminator architecture from SRGAN

4 Datasets

We wanted to specialize MPSRGAN to work on faces, that's why we used portrait data sets. Most of our pictures come from the IDOC Mugshots dataset. 5500 train images, 550 validation and 50 test images were used.

5 Preprocessing

Because our images had different resolutions and aspect ratios, and our model requires 64x64 pixel inputs, we wrote our own image processor. It takes the original image and converts it RGB mode, pads it with black borders to create an aspect ratio of 1:1 and finally resizes the image. It creates our input images and target images for training.

For the generator and discriminator we normalized the pixels from (0, 255) to (-1, 1), but VGG required a different scaling, the subtraction of the mean RGB value from the pixels. It also uses GBR mode instead of RGB.

6 Training

Training a GAN usually takes place in two steps, first training the discriminator, then the generator. We use data loaders to load 16 input and target pictures, the size of one batch. This happens before every iteration. This way we can limit the memory used by the computer. Downside is, we had to use the keras [1] function train on batch and not fit, which caused a lot of problems, mainly we couldn't use callbacks and tensorboard. We used the optimizer adam in both cases with a fixed learning rate of 0.0002 and a beta value of 0.9.

It's usually hard to train a GAN, and hard to identify convergence. When the generator gets better the discriminator has a harder job, and will only have 50 percent success rate. Over time the discriminator loss will get less significant for the generator. It can even hinder the generators progress by not providing useful feedback.

6.1 Discriminator training

The Discriminator takes inputs from both the generator and the high resolution training set. We first have to create a batch of high resolution pictures with the generator. We feed these into the discriminator with a target of zero. Then we take the real high resolution pictures and feed them to the discriminator with a target of one.

The loss function is binary cross entropy. During backpropagation only the discriminator is needed, the generator remains unchanged.

During training it tries to classify the inputs as real or fake, and gets penalized for misclassification.

6.2 Generator training

A more complicated model had to be used for generator training, where we combined the generator, discriminator and VGG in one big network. The generator generates the fake high resolution images from the low resolution inputs. The fake output then feeds into the discriminator and VGG. The former evaluates it, the latter extracts the features from the generated images. The VGG extracts the features form the real high resolution images.

We have two outputs and two losses. One comparison is a binary cross entropy loss between the discriminator output and 1, the desired evaluation for the generator, because it wants to fool the discriminator. The other loss is a mean squared error loss between the extracted features from the generated fake images and the real high resolution images.

Only the generator learns in this phase, the discriminator does not. But we had to include it in the model, because the generator is not directly connected to the loss, because the discriminator produces the output we want to change.

The training ran for a fixed number of epochs.

7 Validation

At every tenth iteration we ran a validation, but we didn't use early stopping, because the losses oscillate a lot during training. Other early stopping methods for GANs were complicated. The best way we found was to look at metrics like peak signal to noise ratio.

8 Testing and Results

For testing we used 50 pictures, and look at the results. Because measuring the quality of image super resolution is hard, we manually evaluated it, to see progress.

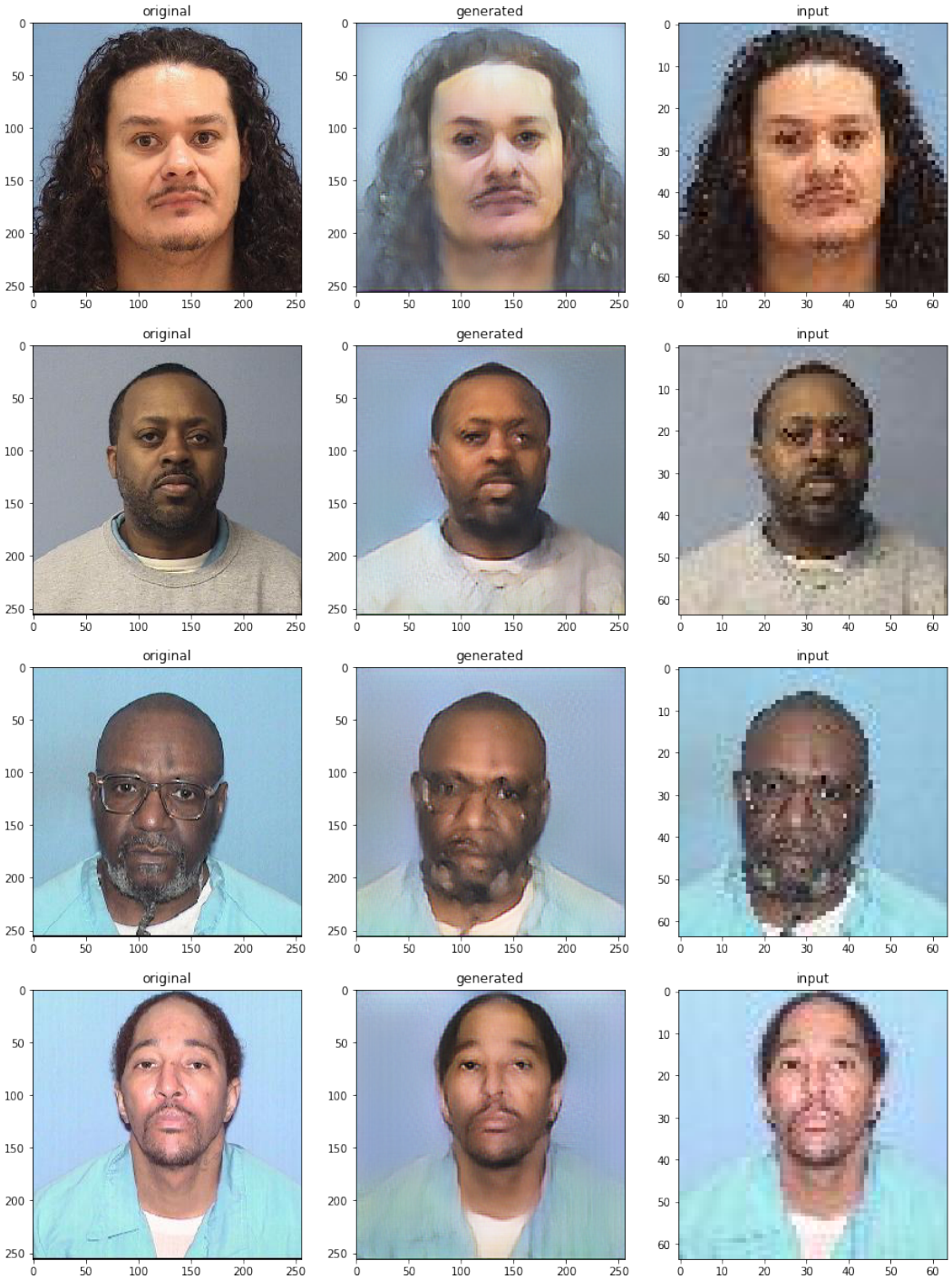

Figure 3: Results on the testing set.

As you can see on the figure above our model is not the state of the art single image super resolution algorithm, it creates overly smooth images, which we wanted to avoid. But it works and with more trining and more data we think we can get better results.

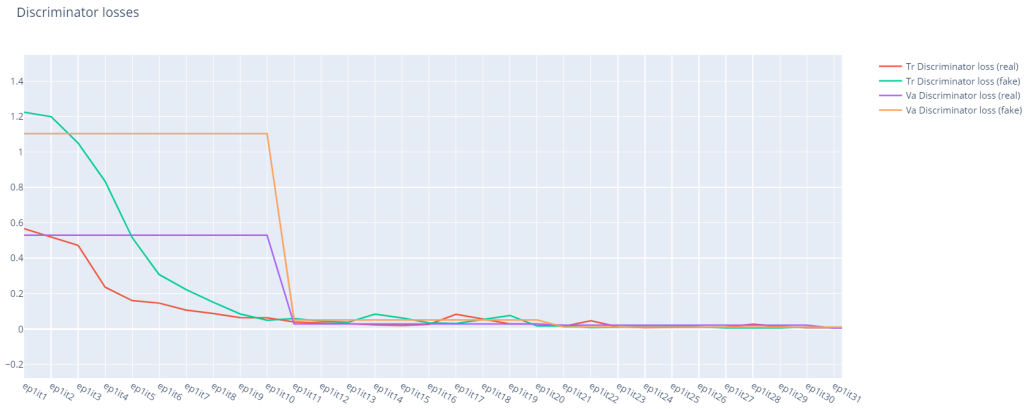

Figure 4: Discriminator loss for training and validation sets, real, and generated

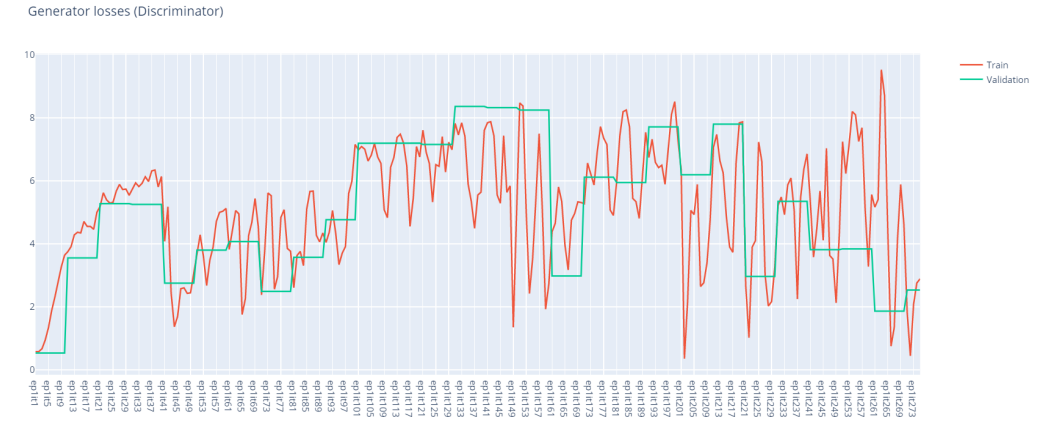

Figure 5: Generator loss from the discriminator

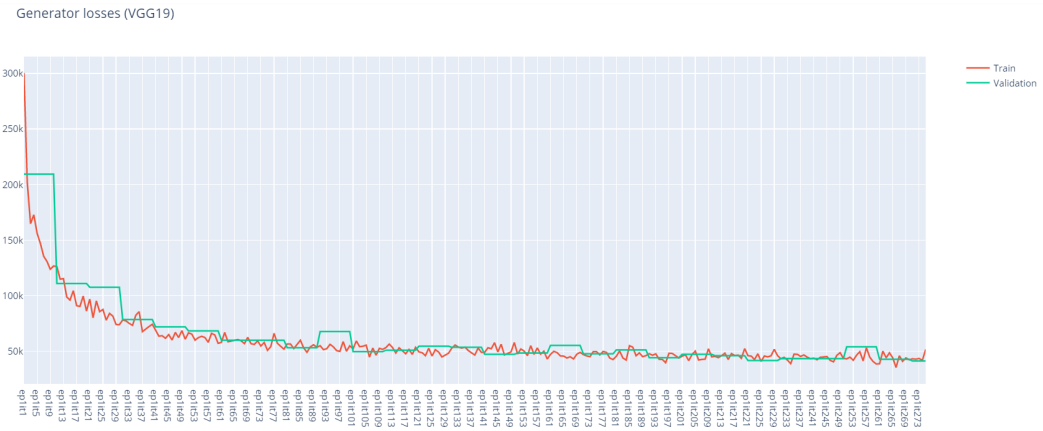

Figure 6: Generator loss from the VGG comparison

On figure 4 the discriminator loss is visible, it shows the losses from the real and generated high resolution images. The discriminator learned easily, but as seen on figure 5 the generator produced easily discoverable images, and the loss oscillated a lot.

The best result was with the VGG loss, it is easy to see, that the generator made images with feature closely matching the originals.

But unfortunately it wasn't enough for convincing results.

9 Conclusion and Future Plans

We didn't achieve the results we were looking for, but in the future we will fix the problems, use a bigger training set and train it for much longer. Adding proper validation and testing and measuring different metrics to evaluate a model is also on our list. Our goal for the future is to make it work on CCTV recordings. For it to work we plan to add multiple angles and cropped images to dataset.

References

[1] François Chollet et al. Keras. `https://keras.io`, 2015.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[5] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29:1153–1160, 12 1981.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 12 1989.

[7] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.