Image Super-Resolution Using Deep Convolutional Networks

1 Citation

Dong, Chao, et al. "Image super-resolution using deep convolutional networks." IEEE transactions on pattern analysis and machine intelligence 38.2 (2016): 295-307.

`https://arxiv.org/pdf/1501.00092v3.pdf`

2 Abstract

We make a Convolutional Neural Network (CNN) that can turn a low resolution image into a high resolution image.

3 Introduction

State of the art super-resolution uses sparse coding. We extract patches from the image, encode them with a low-resolution dictionary, and then turn them into high resolution patches with a high resolution dictionary.

We make Super-Resolution Convolutional Neural Network (SRCNN), a pretty small CNN that sets state of the art on super-resolution and can be trained end-to-end.

4 Related Work

There are four approaches to super-resolution: prediction models, edge based models, image statistical methods, and example based methods. Example based methods are state of the art. Example based methods find similar low resolution patches from the dataset and use their high resolution representation as the output for an input patch.

Most super-resolution work uses grayscale images.

Convolutional Neural Networks have been used for many applications, including de-noising an image. We are the first to make an end-to-end trainable super-resolution model.

5 Convolutional Neural Networks for Super-Resolution

Given a low-resolution image (or image patch), upscale it (with bicubic interpolation) to the desired resolution to get Y. Our network computes $F(Y)$. We hope that $F(Y)$ matches the ground truth high resolution image X. Conceptually, F consists of three pieces: (1) extract patches and turn them into vectors, (2) map the vectors into new vectors in a nonlinear fashion, (3) turn the new vectors into high resolution patches. All three of these steps are done with conv layers.

Here are our three layers, where $*$ represents convolution:

$$F_1(Y) = \max(0, W_1 * Y + B_1)$$
$$F_2(Y) = \max(0, W_2 * F_1(Y) + B_2)$$
$$F(Y) = W_3 * F_2(Y) + B_3$$

where W_1 has n_1 filters of size $c \times f_1, \times f_1$, W_2 has n_2 filters of size $n_1 \times f_2 \times f_2$, and W_3 has c filters of size $n_2 \times f_3 \times f_3$.

Interestingly, the state of the art sparse coding approaches can be viewed as a convolutional neural network as well, but they are not end-to-end trainable like our approach, however. However, leveraging insights from sparse coding, we can set the W_3 filters to be small and we can have $n_2 < n_1$ to encourage compressing representations. Also, to make a decision for a single output pixel, our network considers 169 pixels, while sparse coding only considers 81 pixels.

Our loss function (for parameters $\theta = \{W_1, W_2, W_3, B_1, B_2, B_3\}$) is mean squared error:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \|F(Y_i; \theta) - X_i\|^2$$

Our evaluation metric is Peak signal-to-noise ratio (PSNR), but we also do well on other metrics like structural similarity index metrics (SSIM and MSSIM)

We train with SGD and momentum ($\ell \in \{1, 2, 3\}$ represents the layer):

$$\triangle_{i+1} = 0.9 \cdot \triangle_i - \eta$$
$$W^{\ell}_{i+1} = W^{\ell}_i + \triangle_{i+1}$$

Weights are initialized with a zero-mean Gaussian. Layers 1 and 2 have a learning rate of 10^{-4} while layer 3 has a learning rate of 10^{-5} (this layer needs a lower learning rate in order to converge).

To synthesize data, we take an image, crop out a subimage (ground truth X), subsample the subimage, and upscale it back with bicubic interpolation to get input Y.

We don't have any padding in our CNN (to avoid border effects), so the output is smaller than the input. This means we just evaluate on the central pixels of X.

6 Experiments

We consider different dataset sizes: a small 25K subimage dataset and 5M subimage ImageNet. We validate on the Set5 and Set14 datasets. Although the ImageNet-trained model is better (32.52 dB vs. 32.39 dB), it's a tiny difference. This is probably because the small dataset is pretty representative of natural images and because our neural network is small (8K parameters).

Looking at the filters our model learns, we see things similar to edge detectors, Gaussian/Laplacian pyramids, and texture extractors.

Increasing the number of filters or filter size improves model performance, but causes a large increase in the number of parameters, which slows down training and makes the model more CPU intensive. We tried making our networks deeper, but then they started converging to poor local minima.

We compare against several sparse coding techniques and beat all of them on the PSNR metric. We also produces images that qualitatively look better.

Our model's running time is linear in the resolution of the input image. We are reasonably fast (on a CPU) compared to sparse coding approaches. Our parameters are $(f_1, f_2, f_3, n_1, n_2) = (9, 5, 5, 64, 32)$. Setting $f_2 = 1$ gives a huge speedup, but hurts performance.

Originally, we trained on RGB images, but we also consider YCbCr image channels. Sparse coding only uses the Y channel and uses bicubic interpolation on the other two channel. When we train on all three channels, the results are worse than bicubic interpolation - we get stuck in a bad local minimum. If we pretrain on Y and fine-tune on the other channels, we do worse than if we train on Y alone. So, we conclude that the Cb and Cr channels are not helpful. We just train on RGB. You can alternatively just train on Y if you prefer. Note that PSNR is measured on RGB images.

7 Conclusion

Our SRCNN model does image super-resolution in an end-to-end trainable neural network.