ABSTRACT

Single image super-resolution which aims at recovering a high-resolution image from a single low- resolution image, is a classical problem in computer vision. This problem is inherently ill-posed since a multiplicity of solutions exist for any given low-resolution pixel. In other words, it is an underdetermined in- verse problem, of which solution is not unique. Such a problem is typically mitigated by constraining the solution space by strong prior information. To learn the prior, recent state-of-the-art methods mostly adopt the example-based strategy. Coding-based method is one of the representative external example-based SR methods. This method involves several steps in its solution pipeline. First, overlapping patches are densely cropped from the input image and pre-processed. These patches are then encoded by a low-resolution dictionary. The sparse coefficients are passed into a high-resolution dictionary for reconstructing high-resolution patches. The overlapping constructed patches are aggregated (e.g., by weighted averaging) to produce the final output. This pipeline is shared by most external example-based methods, which pay particular attention to learning and optimizing the dictionaries, or building efficient mapping functions. However, the rest of the steps in the pipeline have been rarely optimized or considered in an unified optimization framework. Our method differs fundamentally from existing external example-based approaches, in that ours does not explicitly learn the dictionaries. We name the proposed model Super-Resolution Convolutional Neural Network the proposed SRCNN has several appealing properties. First, its structure is intentionally designed with simplicity in mind, and yet provides superior accuracy compared with state-of-the-art example-based methods.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

- SIR: Super Image Resolution
- CNN: Convolution Neural Network
- VDSR: very-deep super-resolution (VDSR)

CHAPTER 1

INTRODUCTION

Single image super-resolution which aims at recovering a high-resolution image from a single low- resolution image, is a classical problem in computer vision. This problem is inherently ill-posed since a multiplicity of solutions exist for any given low-resolution pixel. In other words, it is an underdetermined in- verse problem, of which solution is not unique. Such a problem is typically mitigated by constraining the solution space by strong prior information. To learn the prior, recent state-of-the-art methods mostly adopt the example-based strategy. Sparse-coding-based method is one of the representative external example-based SR methods. This method involves several steps in its solution pipeline. First, overlapping patches are densely cropped from the input image and pre-processed. These patches are then encoded by a low-resolution dictionary. The sparse coefficients are passed into a high-resolution dictionary for reconstructing high-resolution patches. The overlapping constructed patches are aggregated (e.g., by weighted averaging) to produce the final output. This pipeline is shared by most external example-based methods, which pay particular attention to learning and optimizing the dictionaries or building efficient mapping functions. However, the rest of the steps in the pipeline have been rarely optimized or considered in an unified optimization framework. Our method differs fundamentally from existing external example-based approaches, in that ours does not explicitly learn the dictionaries. We name the proposed model Super-Resolution Convolutional Neural Network the proposed SRCNN has several appealing properties. First, its structure is intentionally designed with simplicity in mind, and yet provides superior accuracy compared with state-of-the-art example-based methods.

CHAPTER 2

LITERATURE SURVEY

Kim, J., J. K. Lee, and K. M. Lee. "Accurate Image Super-Resolution Using Very Deep Convolutional Networks."

Proceedings of the IEEE® Conference on Computer Vision and Pattern Recognition. 2016, pp. 1646-1654.

Grubinger, M., P. Clough, H. Müller, and T. Deselaers. "The IAPR TC-12 Benchmark: A New Evaluation Resource for Visual Information Systems."

Proceedings of the OntoImage 2006 Language Resources For Content-Based Image Retrieval. Genoa, Italy. Vol. 5, May 2006, p. 10.

He, K., X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification."

Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026-1034.

CHAPTER 3

METHODOLOGY

Several techniques, including deep learning algorithms, have

been proposed to perform SISR.

The VDSR Network

VDSR is a convolutional neural network architecture designed to perform single image super-resolution [1].

The VDSR network learns the mapping between low- and high-resolution images.

This mapping is possible because low-resolution and high-resolution images have similar image content and differ primarily in high-frequency details.

VDSR employs a residual learning strategy, meaning that the network learns to estimate a residual image.

In the context of super-resolution, a residual image is the difference between a high-resolution reference image and a low-resolution image that has been upscaled using bicubic interpolation to match the size of the reference image.

A residual image contains information about the high-frequency details of an image.

The VDSR network detects the residual image from the luminance of a color image.

The luminance channel of an image, Y, represents the brightness of each pixel through a linear combination of the red, green, and blue pixel values.

In contrast, the two chrominance channels of an image, *Cb* and *Cr*, are different linear

combinations of the red, green, and blue pixel values that represent color-difference information.

VDSR is trained using only the luminance channel because human perception is more

sensitive to changes in brightness than to changes in color.

If $Y_{highres}$ is the luminance of the high-resolution image and Y_{lowres} is the luminance a low-resolution image that has been upscaled using bicubic interpolation, then the input to the VDSR network is Y_{lowres} and the network learns to predict $Y_{residual} = Y_{highres} - Y_{lowres}$ from the training data.

After the VDSR network learns to estimate the residual image, you can reconstruct high-resolution images by adding the estimated residual image to the upsampled low-resolution image, then converting the image back to the RGB color space.

A scale factor relates the size of the reference image to the size of the low-resolution image.

As the scale factor increases, SISR becomes more ill-posed because the low-resolution image loses more information about the high-frequency image content. VDSR solves this problem by using a large receptive field.

This example trains a VDSR network with multiple scale factors using scale augmentation.

Scale augmentation improves the results at larger scale factors because the network can take advantage of the image context from smaller scale factors.

Additionally, the VDSR network can generalize to accept images with non-integer scale factors.

Training and Test Data

Download the IAPR TC-12 Benchmark, which consists of 20,000 still natural images [2].

The data set includes photos of people, animals, cities, and more.

- The size of the data file is ~1.8 GB. If you do not want to download the training data set, then you can load the pretrained VDSR network by typing load("trainedVDSRNet.mat"); at the command line.

Then, go directly to the Perform Single Image Super-Resolution Using VDSR Network section in this example.

Use the helper function, downloadIAPRTC12Data, to download the data.

This function is attached to the example as a supporting file.

Specify dataDir as the desired location of the data.

This will train the network with a small subset of the IAPR TC-12 Benchmark data. Load the imageCLEF training data.

All images are 32-bit JPEG color images.

List the number of training images.

Training Data Preperation

To create a training data set, generate pairs of images consisting of upsampled images and the corresponding residual images.

The upsampled images are stored on disk as MAT files in the directory upsampledDirName.

The computed residual images representing the network responses are stored on disk as MAT files in the directory residualDirName.

The MAT files are stored as data type double for greater precision when training the network.

Use the helper function createVDSRTrainingSet to preprocess the training data.

This function is attached to the example as a supporting file.

The helper function performs these operations for each pristine image in trainImages:

- Convert the image to the YCbCr color space
- Downsize the luminance (Y) channel by different scale factors to create sample low-resolution images, then resize the images to the original size using bicubic interpolation.
- Calculate the difference between the pristine and resized images.
- Save the resized and residual images to disk.

Define Pre-processing Pipeline for Training Set

The network inputs are low-resolution images that have been upsampled using bicubic interpolation.

The desired network responses are the residual images.

Create an image datastore called upsampledImages from the collection of input image files.

Create an image datastore called residualImages from the collection of computed residual image files.

Both datastores require a helper function, matRead, to read the image data from the image files.

This function is attached to the example as a supporting file.

Create an image Data Augmenter (Deep Learning Toolbox) that specifies the parameters of data augmentation.

Use data augmentation during training to vary the training data, which effectively

increases the amount of available training data.

Here, the augmenter specifies random rotation by 90 degrees and random reflections in the x-direction.

Create a randomPatchExtractionDatastore that performs randomized patch extraction from the upsampled and residual image datastores.

Patch extraction is the process of extracting a large set of small image patches, or tiles, from a single larger data image.

This type of The resulting datastore, dsTrain, provides mini-batches of data to the network at each iteration of the imageInputLayer epoch. Preview the result of reading from the datastore.

augmentation is frequently used in image-to-image regression problems, where many network architectures can be trained on very small input image sizes.

This means that a large number of patches can be extracted from each full-sized image in the original training set, which greatly increases the size of the training set.

Set Up VDSR Layers

This example defines the VDSR network using 41 individual layers from Deep Learning Toolbox, including:

- (Deep Learning Toolbox) - Image input layer
- convolution2dLayer (Deep Learning Toolbox) - 2-D convolution layer for convolutional neural networks
- reluLayer (Deep Learning Toolbox) - Rectified linear unit (ReLU) layer
- regressionLayer (Deep Learning Toolbox) - Regression output layer for a neural network

Specify Training Options

Train the network using stochastic gradient descent with momentum (SGDM) optimization.

Specify the hyperparameter settings for SGDM by using the trainingOptions (Deep Learning Toolbox) function.

The learning rate is initially 0.1 and decreased by a factor of 10 every 10 epochs.

Train for 100 epochs.

Training a deep network is time-consuming. Accelerate the training by specifying a high learning rate.

However, this can cause the gradients of the network to explode or grow uncontrollably,

preventing the network from training successfully.

To keep the gradients in a meaningful range, enable gradient clipping by specifying "GradientThreshold" as 0.01, and specify "GradientThresholdMethod" to use the L2-norm of the gradients.

Train the Network

By default, the example loads a pretrained version of the VDSR network that has been trained to super-resolve images for scale factors 2, 3 and 4.

The pretrained network enables you to perform super-resolution of test images without waiting for training to complete.

To train the VDSR network, set the doTraining variable in the following code to true.

Train the network using the trainNetwork (Deep Learning Toolbox) function.

Perform Single Image Super-Resolution Using VDSR Network

To perform single image super-resolution (SISR) using the VDSR network, follow the remaining steps of this example:

- Create a sample low-resolution image from a high-resolution reference image.
- Perform SISR on the low-resolution image using bicubic interpolation, a traditional image processing solution that does not rely on deep learning.
- Perform SISR on the low-resolution image using the VDSR neural network.
- Visually compare the reconstructed high-resolution images using bicubic interpolation and VDSR.
- Evaluate the quality of the super-resolved images by quantifying the similarity of the images to the high-resolution reference image.

Create Sample Low-Resolution Image

The test data set, test Images, contains 20 undistorted images shipped in Image Processing Toolbox.

Load the images into an image Datastore and display the images in a montage.

Display the test images as a montage.

Select one of the test images to use for testing the super-resolution network.

Create a low-resolution version of the high-resolution reference image by using imresize with a scaling factor of 0.25.

The high-frequency components of the image are lost during the downscaling.

Improve Image Resolution Using Bicubic Interpolation

A standard way to increase image resolution without deep learning is to use bicubic interpolation.

Upscale the low-resolution image using bicubic interpolation so that the resulting high- resolution image is the same size as the reference image.

Improve Image Resolution Using Pretrained VDSR Network

Recall that VDSR is trained using only the luminance channel of an image because human perception is more sensitive to changes in brightness than to changes in color.

Convert the low-resolution image from the RGB color space to luminance (Iy) and chrominance (Icb and Icr) channels by using the rgb2ycbcr function.

Upscale the luminance and two chrominance channels using bicubic interpolation.

The upsampled chrominance channels, Icb_bicubic and Icr_bicubic, require no further processing.

Pass the upscaled luminance component, Iy_bicubic, through the trained VDSR network.

Observe the activations (Deep Learning Toolbox) from the final layer (a regression layer).

The output of the network is the desired residual image.

Add the residual image to the upscaled luminance component to get the high-resolution VDSR luminance component.

Concatenate the high-resolution VDSR luminance component with the upscaled color components.

Convert the image to the RGB color space by using the ycbcr2rgb function.

The result is the final high-resolution color image using VDSR.

Visual and Quantitative Comparison

To get a better visual understanding of the high-resolution images, examine a small region inside each image.

Specify a region of interest (ROI) using vector roi in the format [x y width height].

The elements define the x- and y-coordinate of the top left corner, and the width and height of the ROI.

Crop the high-resolution images to this ROI, and display the result as a montage.

The VDSR image has clearer details and sharper edges than the high-resolution image created using bicubic interpolation.

Use image quality metrics to quantitatively compare the high-resolution image using bicubic interpolation to the VDSR image.

The reference image is the original high-resolution image, Ireference, before preparing the sample low-resolution image.

Measure the peak signal-to-noise ratio (PSNR) of each image against the reference image.

Larger PSNR values generally indicate better image quality.

See PSNR for more information about this metric.

Measure the structural similarity index (SSIM) of each image.

SSIM assesses the visual impact of three characteristics of an image: luminance, contrast and structure, against a reference image.

The closer the SSIM value is to 1, the better the test image agrees with the reference image.

Bicubic SSIM = 0.9861, vdsr SSIM = 0.9874

Measure perceptual image quality using the Naturalness Image Quality Evaluator (NIQE).

Smaller NIQE scores indicate better perceptual quality.

Calculate the average PSNR and SSIM of the entire set of test images for the scale factors 2, 3, and 4.

For simplicity, you can use the helper function, vdsrMetrics, to compute the average metrics.

This function is attached to the example as a supporting file.

VDSR has better metric scores than bicubic interpolation for each scale factor.

CHAPTER 4

CODING AND TESTING

Download Training and Test Data

- dataDir = tempdir;

  downloadIAPRTC12Data(dataDir)

  ;

- trainImagesDir = fullfile(dataDir,"iaprtc12","images","02");

  exts = [".jpg",".bmp",".png"];

  pristineImages = imageDatastore(trainImagesDir,FileExtensions=exts);

- numel(pristineImages.Files

) ans = 616

- scaleFactors = [2 3 4];

  createVDSRTrainingSet(pristineImages,scaleFactors,upsampledDirName,residualDirName);

Define Preprocessing Pipeline for Training Set

- upsampledImages=imageDatastore(upsampledDirName,FileExtensions=".mat",ReadFcn=@matRead);
  residualImages=imageDatastore(residualDirName,FileExtensions=".mat",ReadFcn=@ma tRead);

- augmenter = imageDataAugmenter(
  ... RandRotatio=@()randi([0,1],1)*90,
  ... RandXReflection=true);

- patchSize = [41 41];
  patchesPerImage =
  64;
  dsTrain = randomPatchExtractionDatastore(upsampledImages,residualImages,patchSize,
  ...
  DataAugmentation=augmenter,PatchesPerImage=patchesPerImage);

- inputBatch =
  preview(dsTrain);
  disp(inputBatch)

InputImage	ResponseImage
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}
{41×41 double}	{41×41 double}

- networkDepth = 20;
  firstLayer = imageInputLayer([41 41 1],Name="InputLayer",Normalization="none");
- convLayer = convolution2dLayer(3,64,Padding=1, ...
  WeightsInitializer="he",BiasInitializer="zeros",Name="Conv1");
- relLayer = reluLayer(Name="ReLU1");
- middleLayers = [convLayer
  relLayer]; for layerNumber =
  2:networkDepth-1
  convLayer = convolution2dLayer(3,64,Padding=[1 1], ...
  WeightsInitializer="he",BiasInitializer="zeros", ...
  Name="Conv"+num2str(layerNumber));
  relLayer =
  reluLayer(Name="ReLU"+num2str(layerNumber));
  middleLayers = [middleLayers convLayer relLayer];
  end
- convLayer = convolution2dLayer(3,1,Padding=[1 1], ...
  WeightsInitializer="he",BiasInitializer="zeros", ...
  NumChannels=64,Name="Conv"+num2str(networkDepth));

- finalLayers = [convLayer regressionLayer(Name="FinalRegressionLayer")];
- layers = [firstLayer middleLayers finalLayers];

Specify Training Options

- maxEpochs = 100;
  epochIntervals = 1;
  initLearningRate = 0.1;
  learningRateFactor =
  0.1; l2reg = 0.0001;
  miniBatchSize = 64;
  options = trainingOptions("sgdm", ...
  Momentum=0.9, ...
  InitialLearnRate=initLearningRate, ...
  LearnRateSchedule="piecewise", ...
  LearnRateDropPeriod=10, ...
  LearnRateDropFactor=learningRateFactor, ...
  L2Regularization=l2reg, ...
  MaxEpochs=maxEpochs, ...
  MiniBatchSize=miniBatchSize, ...
  GradientThresholdMethod="l2norm", ...
  GradientThreshold=0.01, ...
  Plots="training-progress", ...
  Verbose=false);

Train the Network

- doTraining =
  false; if
  doTraining
  net = trainNetwork(dsTrain,layers,options);
  modelDateTime =
  string(datetime("now",Format="yyyy-MM-dd-HH-mm-ss"));
  save("trainedVDSR-"+modelDateTime+".mat","net");

```
else
    load("trainedVDSRNet.mat");
end
```

Create Sample Low-Resolution Image

- ```
  fileNames =
  ["sherlock.jpg","peacock.jpg","fabric.png","greens.jpg", ...
  "hands1.jpg","kobi.png","lighthouse.png","office_4.jpg", ...
  "onion.png","pears.png","yellowlily.jpg","indiancorn.jpg", ...
  "flamingos.jpg","sevilla.jpg","llama.jpg","parkavenue.jpg", ...
  "strawberries.jpg","trailer.jpg","wagon.jpg","football.jpg"];
  filePath =
  fullfile(matlabroot,"toolbox","images","imdata")+filesep;
  filePathNames = strcat(filePath,fileNames);
  testImages = imageDatastore(filePathNames);
  ```

- ```
  montage(testImages)
  ```

- ```
  testImage = "sherlock.jpg";
  Ireference = imread(testImage);
  Ireference =
  im2double(Ireference);
  imshow(Ireference)
  title("High-Resolution Reference Image")
  ```

- ```
  scaleFactor = 0.25;
  Ilowres = imresize(Ireference,scaleFactor,"bicubic");
  imshow(Ilowres)
  title("Low-Resolution Image")
  ```

Improve Image Resolution Using Bicubic Interpolation

- ```
  [nrows,ncols,np] = size(Ireference);
  Ibicubic = imresize(Ilowres,[nrows
  ncols],"bicubic"); imshow(Ibicubic)
  title("High-Resolution Image Obtained Using Bicubic Interpolation")
  ```
  ```

Improve Image Resolution Using Pretrained VDSR Network

- Iycbcr =
  rgb2ycbcr(Ilowres); Iy =
  Iycbcr(:,:,1);
  Icb = Iycbcr(:,:,2);
  Icr = Iycbcr(:,:,3);

- Iy_bicubic = imresize(Iy,[nrows ncols],"bicubic");
  Icb_bicubic = imresize(Icb,[nrows
  ncols],"bicubic"); Icr_bicubic = imresize(Icr,[nrows
  ncols],"bicubic");

- Iresidual =
  activations(net,Iy_bicubic,41); Iresidual
  = double(Iresidual); imshow(Iresidual,[])
  title("Residual Image from VDSR")

- Isr = Iy_bicubic + Iresidual;

- Ivdsr = ycbcr2rgb(cat(3,Isr,Icb_bicubic,Icr_bicubic));
  imshow(Ivdsr)
  title("High-Resolution Image Obtained Using VDSR")

Visual and Quantitative Comparison

- roi = [360 50 400 350];

- montage({imcrop(Ibicubic,roi),i
  mcrop(Ivdsr,roi)})
  title("High-Resolution Results Using Bicubic Interpolation (Left) vs. VDSR (Right)");
- bicubicPSNR = psnr(Ibicubic,Ireference)
- vdsrPSNR = psnr(Ivdsr,Ireference)
- bicubicSSIM = ssim(Ibicubic,Ireference)
- vdsrSSIM = ssim(Ivdsr,Ireference)
- bicubicNIQE = niqe(Ibicubic)
- vdsrNIQE = niqe(Ivdsr)
- scaleFactors = [2 3 4];
  vdsrMetrics(net,testImages,scaleFactors)
  ;

CHAPTER 5

SCREENSHOTS AND RESULTS

Fig 1

Fig 2

Fig 3

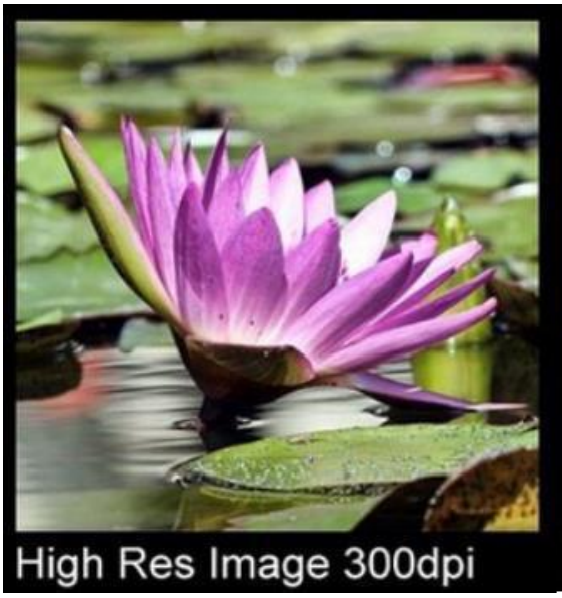

Fig 4

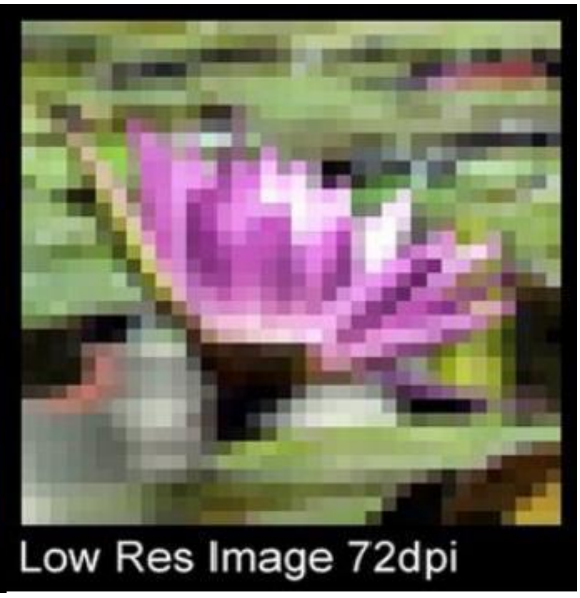

Fig 5

- bicubicPSNR = 38.4747
- vdsrPSNR = 39.2346
- bicubicSSIM = 0.9861
- vdsrSSIM = 0.9874
- Results for Scale factor 2
- Average PSNR for Bicubic = 31.467070
- Average PSNR for VDSR = 31.481973
- Average SSIM for Bicubic = 0.935820
- Average SSIM for VDSR = 0.947057
- Results for Scale factor 3
- Average PSNR for Bicubic = 28.107057
- Average PSNR for VDSR = 28.430546
- Average SSIM for Bicubic = 0.883927
- Average SSIM for VDSR = 0.894634
- Results for Scale factor 4
- Average PSNR for Bicubic = 27.066129
- Average PSNR for VDSR = 27.846590
- Average SSIM for Bicubic = 0.863270
- Average SSIM for VDSR = 0.878101

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

- VDSR has better metric scores than bicubic interpolation for each scale factor.
- Conventional sparse-coding-based SR methods can be reformulated into a deep convolutional neural network.
- The proposed approach, SRCNN, learns an end-to-end mapping between low- and high- resolution images, with little extra pre/post-processing beyond the optimization.
- With a lightweight structure, the SRCNN has achieved superior performance than the state- of-the-art methods.
- We conjecture that additional performance can be further gained by exploring more filters and different training strategies.
- One could also investigate a network to cope with different upscaling factors.

REFERENCES

https://www.analyticsvidhya.com/blog/2021/05/deep-learning-for-image-super-resolution/

https://blog.paperspace.com/image-super-resolution/

https://in.mathworks.com/help/images/single-image-super-resolution-using-deep
- learning.html

Image Super-Resolution Using Deep Convolutional Networks Chao Dong, Chen Change Loy, Member, IEEE, Kaiming He, Member, IEEE, and Xiaoou Tang, Fellow, IEEE.