A Project Report

On

IMAGE SUPER-RESOLUTION USING DEEPCONVOLUTION NETWORKS.

Submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Mr. S.PRAKASH
Assistant Professor

Submitted By:

SACHIN CHAUHAN 19SCSE1010107

SAMAR MAHMOOD 19SCSE1010195

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
GALGOTIAS UNIVERSITY, GREATER NOIDA,INDIA

December-2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project, entitled **"IMAGE SUPER-RESOLUTION USING DEEP CONVOLUTION NETWORKS."** in partial fulfillment of the requirements for the award of the B-Tech CSE submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of July, 2021 to December, 2021, under the supervision of Mr.S.Prakash, Assistant Professor, Department of Computer Science and Engineering, of School of Computing Science and Engineering , Galgotias University, Greater Noida, India.

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other places.

Sachin Chauhan 19SCSE1010107

Samar Mahmood 19SCSE1010195

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr. S.Prakash
Assistant Professor

CERTIFICATE

The Final Project Viva-Voce examination of Sachin Chauhan: 19SCSE1010107 and Samar Mahmood: 19SCSE1010195 has been held on _____ and his work is recommended for the award of B.Tech-CSE.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: 24 December 2021
Place: Greater Noida

ACKNOWLEDGEMENT

I would like to thank my guide Mr S.Prakash who gave me this opportunity to work on this project. I got to learn a lot from this project about Image super-resolution using deep convolution networks.

At last, I would like to extend my heartfelt thanks to my guide because without their help this project would not have been successfully completed. Finally, I would like to thank my colleagues who have been with me all the time.

ABSTRACT

We are using deep learning methods for single image super-resolution. In this model our system will directly learns an end-to-end mapping between low and high resolutions images. And for mapping we are using deep convolution neural network (CNN) that will take low - resolution images as an input and gives an high resolution output image. We further show that traditional sparse- coding-based SR methods can also be viewed as a deep convolutional network. But unlike traditional methods that handle each component separately, our method jointly optimizes all layers. Our deep CNN has a lightweight structure, yet demonstrates state- of-the-art restoration quality, and achieves fast speed for practical on-line usage. We explore different network structures and parameter settings to achieve trade-offs between performance and speed. Moreover, we extend our network to cope with three color channels simultaneously, and show better overall reconstruction quality.

CONTENTS

CHAPTER-1
INTRODUCTION

1.1 Basic Introduction

Image super-resolution is the task of recovering a high-resolution image from a lower-resolution image. This problem is notable for itsapplications in security as well as in medical imaging, especially since image reconstruction offers a methodology for correcting imaging system imperfections. For our project, we implement SRCNN and refine the model in order to improve the quality of the output images, as measured by peak signal-to-noise ratio (PSNR). The input to our algorithm is a low-resolution image, which we feed through a convolutional neural network (CNN) in order to produce ahigh-resolution image. Traditional methods for image up sampling rely on low-information, smooth interpolation between known pixels. Such methods can be treated as a convolution with a kernel encoding no information about the original photograph. Although they increase the resolution of an image, they fail to produce the clarity desired in the super-resolution task. Convolutional Neural Networks (CNNs) are a generalization of such algorithms, using learned kernels with nonlinear activations to encode general characteristics about photographs that can add structure lost in the low-resolution input. CNN architectures such as SRCNN have been successfully applied to the super-resolution task.

1.2 Image Super-Resolution

According to the image priors, single-image super resolution algorithms can be

categorized into four types – prediction models, edge based methods, image statistical methods and patch based (or example-based) methods. These methods have been thoroughly investigated and evaluated in Yang et al.'s work . Among them, the example-based methods achieve the state-of-the-art performance. The internal example-based methods exploit the selfsimilarity property and generate exemplar patches from the input image. It is first proposed in Glasner's work , and several improved variants are roposed to accelerate the implementation. The external example-based methods learn a mapping between low/highresolution patches from external datasets. These studies vary on how to learn a compact dictionary or manifold space to relate low/high-resolution patches, and on how representation schemes can be conducted in such spaces. In the pioneer work of Freeman et al. , the dictionaries are directly presented as low/high-resolution patch pairs, and the nearest neighbour (NN) of the input patch is found in the low-resolution space, with its corresponding high-resolution patch used for reconstruction. Chang et al. introduce a manifold embedding technique as an alternative to the NN strategy. In Yang et al.'s work , the above NN correspondence advances to a more sophisticated sparse coding formulation. Other mapping functions such as kernel regression , simple 3 function, random forest and anchored neighborhood regression are proposed to further improve the mapping accuracy and speed. The sparsecoding-based method and its several improvements are among the state-of-the-art SR methods nowadays. In these methods, the patches are the focus of the optimization; the patch extraction and aggregation steps are considered as

pre/post-processing and handled separately. The majority of SR algorithms focus on gray-scale or single-channel image super-resolution. For color images, the aforementioned methods first transform the problem to a different color space (YCbCr or YUV), and SR is applied only on the luminance channel. There are also works attempting to super-resolve all channels simultaneously. For example, Kim and Kwon and Dai et al. apply their model to each RGB channel and combined them to produce the final results. However, none of them has analyzed the SR performance of different channels, and the necessity of recovering all three channels.

1.3 Convolutional Neural Networks

Convolutional neural networks (CNN) date back decades and deep CNNs have recently shown an explosive popularity partially due to its success in image classification . They have also been successfully applied to other computer vision fields, such as object detection , face recognition , and pedestrian detection . Several factors are of central importance in this progress: (i) the efficient training implementation on modern powerful GPUs , (ii) the proposal of the Rectified Linear Unit (ReLU) which makes convergence much faster while still presents good quality , and (iii) the easy access to an abundance of data (like ImageNet) for training larger models. Our method also benefits from these progresses.

1.4 Deep Learning for Image Restoration

There have been a few studies of using deep learning techniques for image restoration. The multi-layer perceptron (MLP), whose all layers are fully-connected (in contrast to convolutional), is applied for natural image denoising and post-deblurring denoising . More closely related to our work, the convolutional neural network is applied for natural image denoising and removing noisy patterns (dirt/rain) . These restoration problems are more or less denoising-driven. Cui et al. propose to embed auto-encoder networks in their superresolution pipeline under the notion internal examplebased approach . The deep model is not specifically designed to be an end-to-end solution, since each layer of the cascade requires independent optimization of the self-similarity search process and the auto-encoder. On the contrary, the proposed SRCNN optimizes an end-toend mapping. Further, the SRCNN is faster at speed. It is not only a quantitatively superior method, but also a practically useful one.

CHAPTER-2
FUNCTIONALITY & WORKING

2.1 Formulation

Consider a single low-resolution image, we first upscale it to the desired size using bicubic interpolation, which is the only pre-processing we perform3. Let us denote the interpolated image as Y. Our goal is to recover from Y an image F(Y) that is as similar as possible to the ground truth high-resolution image X. For the ease of presentation, we still call Y a “low-resolution” image, although it has the same size as X. We wish to learn a mapping F, which conceptually consists of three operations:

1) **Patch extraction and representation:** this operation extracts (overlapping) patches from the lowresolution image Y and represents each patch as a high-dimensional vector. These vectors comprise a set of feature maps, of which the number equals tothe dimensionality of the vectors.

2) **Non-linear mapping:** this operation nonlinearly maps each high-dimensional vector onto another high-dimensional vector. Each mapped vector is conceptually the representation of a high-resolution patch. These vectors comprise another set of feature maps.

3) **Reconstruction:** this operation aggregates the above high-resolution patch-wise representations to generate the final high-resolution image. This image is expected to be similar to the ground truth X.

We will show that all these operations form a convolutional neural network. An

overview of the network is depicted in Figure 2. Next we detail our definition of each operation.

2.1.1 Patch extraction and representation

A popular strategy in image restoration is to densely extract patches and then represent them by a set of pre-trained bases such as PCA, DCT, Haar, etc. This is equivalent to convolving the image by a set of filters, each of which is a basis. In our formulation, we involve the optimization of these bases into the optimization of the network. Formally, our first layer is expressed as an operation F1:

$$F1(Y) = \max (0; W1 * Y + B1) ;$$

where W1 and B1 represent the filters and biases respectively, and '*' denotes the convolution operation. Here, W1 corresponds to n1 filters of support c x f1 x f1, where c is the number of channels in the input image, f1 is the spatial size of a filter. Intuitively, W1 applies n1 convolutions on the image, and each convolution has a kernel size c * f1 * f1. The output is composed of n1 feature maps. B1 is an n1-dimensional vector, whose each element is associated with a filter. We apply the Rectified Linear Unit (ReLU, max(0; x)) on the filter responses.

2.1.2 Non-linear mapping

The first layer extracts an n1-dimensional feature for each patch. In the second operation, we map each of these n1-dimensional vectors into an n2-dimensional one. This is equivalent to applying n2 filters which have a trivial spatial support 1*1. This interpretation is only valid for 1*1 filters. But it is easy to generalize to larger filters like 3*3 or 5*5. In that case, the non-linear mapping is not on a patch

of the input image; instead, it is on a 3*3 or 5*5 “patch” of the feature map. The operation of the second layer is:

$$F2(Y) = \max (0;W2 * F1(Y) + B2) .$$

Here W2 contains n2 filters of size n1 x f2 x f2, and B2 is n2-dimensional. Each of the output n2-dimensional vectors is conceptually a representation of a high-resolution patch that will be used for reconstruction. It is possible to add more convolutional layers to increase the non-linearity. But this can increase the complexity of the model (n2 x f2 x f2 x n2 parameters for one layer), and thus demands more training time.

2.1.3 Reconstruction

In the traditional methods, the predicted overlapping high-resolution patches are often averaged to produce the final full image. The averaging can be considered as a pre-defined filter on a set of feature maps (where each position is the “flattened” vector form of a highresolution patch). Motivated by this, we define a convolutional layer to produce the final high-resolution image:

$$F(Y) = W3* F2(Y) + B3$$

Here W3 corresponds to c filters of a size n2 x f3 x f3, and B3 is a c-dimensional vector. If the representations of the high-resolution patches are in the image domain (i.e.,we can simply reshape each representation to form the patch), we expect that the filters act like an averaging filter; if the representations of the high-resolution patches are in some other domains (e.g.,coefficients in terms of some bases), we expect that W3 behaves like first projecting the coefficients onto the image domain

and then averaging. In either way, W3 is a set of linear filters. Interestingly, although the above three operations are motivated by different intuitions, they all lead to thesame form as a convolutional layer. We put all three operations together and form a convolutional neural network (Figure 2). In this model, all the filtering weights and biases are to be optimized. Despite the succinctness of the overall structure, our SRCNN model is carefully developed by drawing extensive experience resulted from significant progresses in super-resolution [49], [50]. We detail the relationship in the next section.

2.2 Relationship to Sparse-Coding-Based Methods

We show that the sparse-coding-based SR methods [49], [50] can be viewed as a convolutional neural network. Figure 3 shows an illustration. In the sparse-coding-based methods, let us consider that an f1 x f1 low-resolution patch is extracted from the input image. Then the sparse coding solver, like Feature-Sign [29], will first project the patch onto a (lowresolution) dictionary. If the dictionary size is n1, this is equivalent to applying n1 linear filters (f1 x f1) on the input image (the mean subtraction is also a linear operation so can be absorbed). This is illustrated as the left part of Figure 3. The sparse coding solver will then iteratively process the n1 coefficients. The outputs of this solver are n2 coefficients, and usually n2 = n1 in the case of sparse coding. These n2 coefficients are the representation of the high-resolution patch. In this sense, the sparse coding solver behaves as a special case of a non-linear mappingoperator, whose spatial support is 1 x 1. See the middle part of Figure 3. However, the sparse coding solver is not feed-forward, i.e.,it is an

iterative algorithm. On the contrary, our non-linear operator is fully feed-forward and can be computed efficiently. If we set f2 = 1, then our non-linear operator can be considered as a pixel-wise ully-connected layer. It is worth noting that “the sparse coding solver” in SRCNN refers to the first two layers, but not just the second layer or the activation function (ReLU). Thus the nonlinear operation in SRCNN is also well optimized through the learning process. The above n2 coefficients (after sparse coding) are then projected onto another (high-resolution) dictionary to produce a high-resolution patch. The overlapping high-resolution patches are then averaged. As discussed bove, this is equivalent to linear convolutions on the n2 feature maps. If the high-resolution patches used for reconstruction are of size f3 x f3, then the linear filters have an equivalent spatial support of size f3 x f3. See the right part of Figure 3. The above discussion shows that the sparse-codingbased SR method can be viewed as a kind of convolutional neural network (with a different non-linear mapping). But not all operations have been considered in the optimization in the sparse-coding-based SR methods. On the contrary, in our convolutional neural network, the low-resolution dictionary, high-resolution dictionary, non-linear mapping, together with mean subtraction and averaging, are all involved in the filters to be optimized. So our method optimizes an end-to-end mapping that consists of all operations. The above analogy can also help us to design hyperparameters. For example, we can set the filter size ofthe last layer to be smaller than that of the first layer, and thus we rely more on the central part of the highresolution patch (to the extreme, if f3 = 1, we are using the center

pixel with no averaging). We can also set n2 < n1 because it is expected to be sparser. A typical and basic setting is f1 = 9, f2 = 1, f3 = 5, n1 = 64, and n2 = 32 (we evaluate more settings in the experiment section). On the whole, the estimation of a high resolution pixel utilizes the information of (9 + 5 - 1)2 = 169 pixels. Clearly, the information exploited for reconstruction is comparatively larger than that used in existing external example-based approaches, e.g., using (5+5-1)2 = 81 pixels5 [15], [50]. This is one of the reasons why the SRCNN gives superior performance.

CHAPTER-3
REQUIREMENTS

3.1 Requirements

It should be noted that tensorflow version matters a lot because old versions don't include some layers such as depth-to-space, so you should make sure tf version is larger than 2.4.0. Another important thing is that only tf-nightly larger than 2.5.0 can perform arbitrary input shape quantization. I provide two conda environments, tf.yaml for training and tfnightly.yaml for Post-Training Quantization(PTQ) and Quantization-Aware Training(QAT). You can use the following scripts to create two separate conda environments.

```
conda env create -f tf.yaml
```

```
conda env create -f tfnightly.yaml
```

3.2 Pipeline

1. Train and validate on DIV2K. We can achieve 30.22dB with 42.54K parameters.
2. Post-Training Quantization: after int8 quantization, PSNR drops to 30.09dB.
3. Quantization-Aware Training: Insert fake quantization nodes during training. PSNR increases to 30.15dB, which means the model size becomes 4x smaller with only 0.07dB performance loss.

3.3 Prepare DIV2K Data

Download DIV2K and put DIV2K in data folder. Then the structure should look like:

```
| DATA
|       | DIV2K
|       |       |DIV2K_train_HR
|       |       |       | 0001.png
|       |       |       |……
|       |       |       | 0900.png
|       |       |DIV2K_train_LR_bicubic
|       |       |       |X2
|       |       |       |       |0001x2.png
|       |       |       |       |…
|       |       |       |       |0900x2.png
```

3.4 Training

python train.py --opt options/train/base7.yaml --name ase7_D4C28_bs16ps64_lr1e-3 --scale 3 --bs 16 --ps 64 --lr 1e-3 --gpu_ids 0

Note: The argument --name specifies the following save path:

- Log file will be saved in log/{name}.log
- Checkpoint and current best weights will be saved in experiment/{name}/best_status/
- Visualization of Train and Validate will be saved in Tensorboard/{name}/

You can use tensorboard to monitor the training and validating process by:

tensorboard --logdir Tensorboard

3.5 Quantization-Aware Training

If you haven't worked with Tensorflow Lite and network quantization before, please refer to official guideline. This technology inserts fake quantization nodes to make

the weights aware that themselves will be quantized. For this model, you can simply use the following script to perform QAT:

python train.py --opt options/train/base7_qat.yaml --name base7_D4C28_bs16ps64_lr1e-3_qat --scale 3 --bs 16 --ps 64 --lr 1e-3 --gpu_ids 0 --qat --qat_path experiment/base7_D4C28_bs16ps64_lr1e-3/best_status

Convert to TFLite which can run on mobile device

python generate_tflite.py

Then the converted tflite model will be saved in TFMODEL/. TFMODEL/{name}.tflite is used for predicting high-resolution image(arbitary low-resolution input shape is allowed), while TFMODEL/{name}_time.tflite fixes model input shape to [1, 360, 640, 3] for getting inference time.

3.6 Run TFLite Model on your own devices

Download AI Benchmark from the Google Play / website and run its standard tests. After the end of the tests, enter the PRO Model and select the Custom Model tab there.Send your tflite model to your device and remember its location, then run the model.

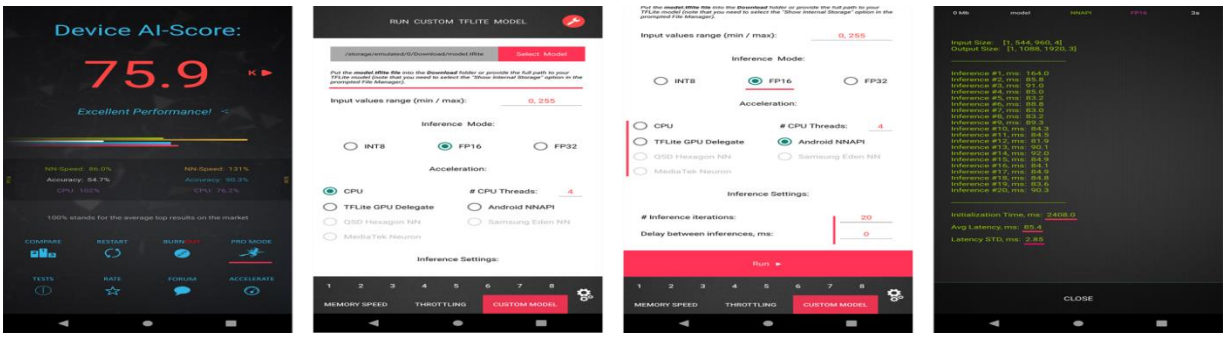


Fig : Loading and running custom TensorFlow Lite models with AI Benchmark application

CHAPTER-4
IMPLEMENTATION

Implementation

```
# check package versions
import sys
import keras
import cv2
import numpy
import matplotlib
import skimage

print('Python: {}'.format(sys.version))
print('Keras: {}'.format(keras.__version__))
print('OpenCV: {}'.format(cv2.__version__))
print('NumPy: {}'.format(numpy.__version__))
print('Matplotlib: {}'.format(matplotlib.__version__))
print('Scikit-Image: {}'.format(skimage.__version__))
```

```
Python: 2.7.13 |Continuum Analytics, Inc.| [MSC v.1500 64 bit (AMD64)]
Keras: 2.1.4
OpenCV: 3.3.0
NumPy: 1.14.1
Matplotlib: 2.1.0
Scikit-Image: 0.13.1
```

In [4]:

```
# import the necessary packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.optimizers import Adam
from skimage.measure import compare_ssim as ssim
from matplotlib import pyplot as plt
import cv2
import numpy as np
import math
import os

# python magic function, displays pyplot figures in the notebook
%matplotlib inline
# define a function for peak signal-to-noise ratio (PSNR)
```

```
def psnr(target, ref):

    # assume RGB image
    target_data = target.astype(float)
    ref_data = ref.astype(float)

    diff = ref_data - target_data
    diff = diff.flatten('C')

    rmse = math.sqrt(np.mean(diff ** 2.))

    return 20 * math.log10(255. / rmse)

# define function for mean squared error (MSE)
def mse(target, ref):
    # the MSE between the two images is the sum of the squared difference between the two
images
    err = np.sum((target.astype('float') - ref.astype('float')) ** 2)
    err /= float(target.shape[0] * target.shape[1])

    return err

# define function that combines all three image quality metrics
def compare_images(target, ref):
    scores = []
    scores.append(psnr(target, ref))
    scores.append(mse(target, ref))
    scores.append(ssim(target, ref, multichannel =True))

    return scores
# prepare degraded images by introducing quality distortions via resizing

def prepare_images(path, factor):

    # loop through the files in the directory
    for file in os.listdir(path):

        # open the file
        img = cv2.imread(path + '/' + file)

        # find old and new image dimensions
        h, w, _ = img.shape
        new_height = h / factor
        new_width = w / factor
```

```
        # resize the image - down
        img = cv2.resize(img, (new_width, new_height), interpolation = cv2.INTER_LINEAR)

        # resize the image - up
        img = cv2.resize(img, (w, h), interpolation = cv2.INTER_LINEAR)

        # save the image
        print('Saving {}'.format(file))
        cv2.imwrite('images/{}'.format(file), img)
prepare_images('source/', 2)
```

```
Saving baboon.bmp
Saving baby_GT.bmp
Saving barbara.bmp
Saving bird_GT.bmp
Saving butterfly_GT.bmp
Saving coastguard.bmp
Saving comic.bmp
Saving face.bmp
Saving flowers.bmp
Saving foreman.bmp
Saving head_GT.bmp
Saving lenna.bmp
Saving monarch.bmp
Saving pepper.bmp
Saving ppt3.bmp
Saving woman_GT.bmp
Saving zebra.bmp
```

```
# test the generated images using the image quality metrics

for file in os.listdir('images/'):

    # open target and reference images
    target = cv2.imread('images/{}'.format(file))
    ref = cv2.imread('source/{}'.format(file))

    # calculate score
    scores = compare_images(target, ref)

    # print all three scores with new line characters (\n)
    print('{}\nPSNR: {}\nMSE: {}\nSSIM: {}\n'.format(file, scores[0], scores[1], scores[2]))
```

```
baboon.bmp
```

PSNR: 22.1570840834
MSE: 1187.11613333
SSIM: 0.6292775879

baby_GT.bmp
PSNR: 34.3718064097
MSE: 71.2887458801
SSIM: 0.935698787272

barbara.bmp
PSNR: 25.9066298376
MSE: 500.655085359
SSIM: 0.809863264641

bird_GT.bmp
PSNR: 32.8966447287
MSE: 100.123758198
SSIM: 0.953364486603

butterfly_GT.bmp
PSNR: 24.7820765603
MSE: 648.625411987
SSIM: 0.879134476384

coastguard.bmp
PSNR: 27.1616006639
MSE: 375.008877841
SSIM: 0.756950063355

comic.bmp
PSNR: 23.7998615022
MSE: 813.233883657
SSIM: 0.83473354164

face.bmp
PSNR: 30.9922065029
MSE: 155.231897185
SSIM: 0.800843949229

flowers.bmp
PSNR: 27.4545048054
MSE: 350.550939227
SSIM: 0.869728628697

foreman.bmp
PSNR: 30.1445653266
MSE: 188.688348327
SSIM: 0.933268417389

head_GT.bmp
PSNR: 31.0205028482
MSE: 154.22377551
SSIM: 0.801112133073

lenna.bmp
PSNR: 31.4734929787
MSE: 138.948005676
SSIM: 0.846098920052

monarch.bmp
PSNR: 30.1962423653
MSE: 186.456436157
SSIM: 0.943957429343

pepper.bmp
PSNR: 29.8894716169
MSE: 200.103393555
SSIM: 0.835793756846

ppt3.bmp
PSNR: 24.8492616895
MSE: 638.668426391
SSIM: 0.928402394232

woman_GT.bmp
PSNR: 29.3262362808
MSE: 227.812729498
SSIM: 0.933539728047

zebra.bmp
PSNR: 27.9098406393
MSE: 315.658545953
SSIM: 0.891165620933

```
# define the SRCNN model
def model():

    # define model type
```

```
    SRCNN = Sequential()

    # add model layers
    SRCNN.add(Conv2D(filters=128, kernel_size = (9, 9), kernel_initializer='glorot_uniform',
                     activation='relu', padding='valid', use_bias=True, input_shape=(None, None, 1)))
    SRCNN.add(Conv2D(filters=64, kernel_size = (3, 3), kernel_initializer='glorot_uniform',
                     activation='relu', padding='same', use_bias=True))
    SRCNN.add(Conv2D(filters=1, kernel_size = (5, 5), kernel_initializer='glorot_uniform',
                     activation='linear', padding='valid', use_bias=True))

    # define optimizer
    adam = Adam(lr=0.0003)

    # compile model
    SRCNN.compile(optimizer=adam, loss='mean_squared_error',
metrics=['mean_squared_error'])

    return SRCNN
# define necessary image processing functions

def modcrop(img, scale):
    tmpsz = img.shape
    sz = tmpsz[0:2]
    sz = sz - np.mod(sz, scale)
    img = img[0:sz[0], 1:sz[1]]
    return img


def shave(image, border):
    img = image[border: -border, border: -border]
    return img
```

In [11]:

```
# define main prediction function

def predict(image_path):

    # load the srcnn model with weights
    srcnn = model()
    srcnn.load_weights('3051crop_weight_200.h5')

    # load the degraded and reference images
    path, file = os.path.split(image_path)
    degraded = cv2.imread(image_path)
    ref = cv2.imread('source/{}'.format(file))
```

```
    # preprocess the image with modcrop
    ref = modcrop(ref, 3)
    degraded = modcrop(degraded, 3)

    # convert the image to YCrCb - (srcnn trained on Y channel)
    temp = cv2.cvtColor(degraded, cv2.COLOR_BGR2YCrCb)

    # create image slice and normalize
    Y = numpy.zeros((1, temp.shape[0], temp.shape[1], 1), dtype=float)
    Y[0, :, :, 0] = temp[:, :, 0].astype(float) / 255

    # perform super-resolution with srcnn
    pre = srcnn.predict(Y, batch_size=1)

    # post-process output
    pre *= 255
    pre[pre[:] > 255] = 255
    pre[pre[:] < 0] = 0
    pre = pre.astype(np.uint8)

    # copy Y channel back to image and convert to BGR
    temp = shave(temp, 6)
    temp[:, :, 0] = pre[0, :, :, 0]
    output = cv2.cvtColor(temp, cv2.COLOR_YCrCb2BGR)

    # remove border from reference and degraged image
    ref = shave(ref.astype(np.uint8), 6)
    degraded = shave(degraded.astype(np.uint8), 6)

    # image quality calculations
    scores = []
    scores.append(compare_images(degraded, ref))
    scores.append(compare_images(output, ref))

    # return images and scores
    return ref, degraded, output, scores
ref, degraded, output, scores = predict('images/flowers.bmp')

# print all scores for all images
print('Degraded Image: \nPSNR: {}\nMSE: {}\nSSIM: {}\n'.format(scores[0][0],
scores[0][1], scores[0][2]))
print('Reconstructed Image: \nPSNR: {}\nMSE: {}\nSSIM: {}\n'.format(scores[1][0],
scores[1][1], scores[1][2]))
```

```
# display images as subplots
fig, axs = plt.subplots(1, 3, figsize=(20, 8))
axs[0].imshow(cv2.cvtColor(ref, cv2.COLOR_BGR2RGB))
axs[0].set_title('Original')
axs[1].imshow(cv2.cvtColor(degraded, cv2.COLOR_BGR2RGB))
axs[1].set_title('Degraded')
axs[2].imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
axs[2].set_title('SRCNN')

# remove the x and y ticks
for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])
```

Degraded Image:

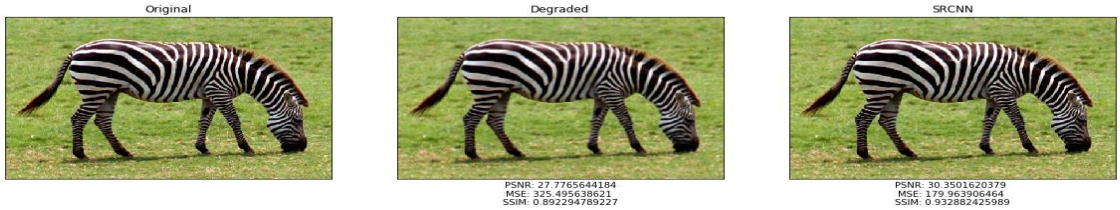
PSNR: 27.2486864596
MSE: 367.564000474
SSIM: 0.86906220246

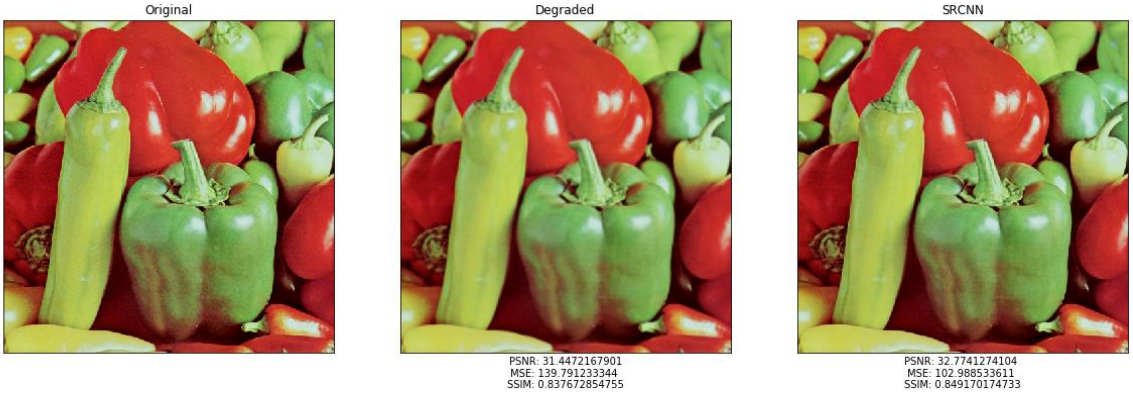
Reconstructed Image:
PSNR: 29.6675381755
MSE: 210.594874985
SSIM: 0.899043290319

```
for file in os.listdir('images'):

    # perform super-resolution
    ref, degraded, output, scores = predict('images/{}'.format(file))

    # display images as subplots
    fig, axs = plt.subplots(1, 3, figsize=(20, 8))
    axs[0].imshow(cv2.cvtColor(ref, cv2.COLOR_BGR2RGB))
    axs[0].set_title('Original')
    axs[1].imshow(cv2.cvtColor(degraded, cv2.COLOR_BGR2RGB))
    axs[1].set_title('Degraded')
    axs[1].set(xlabel = 'PSNR: {}\nMSE: {} \nSSIM: {}'.format(scores[0][0], scores[0][1],
scores[0][2]))
    axs[2].imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
    axs[2].set_title('SRCNN')
    axs[2].set(xlabel = 'PSNR: {} \nMSE: {} \nSSIM: {}'.format(scores[1][0], scores[1][1],
scores[1][2]))
```

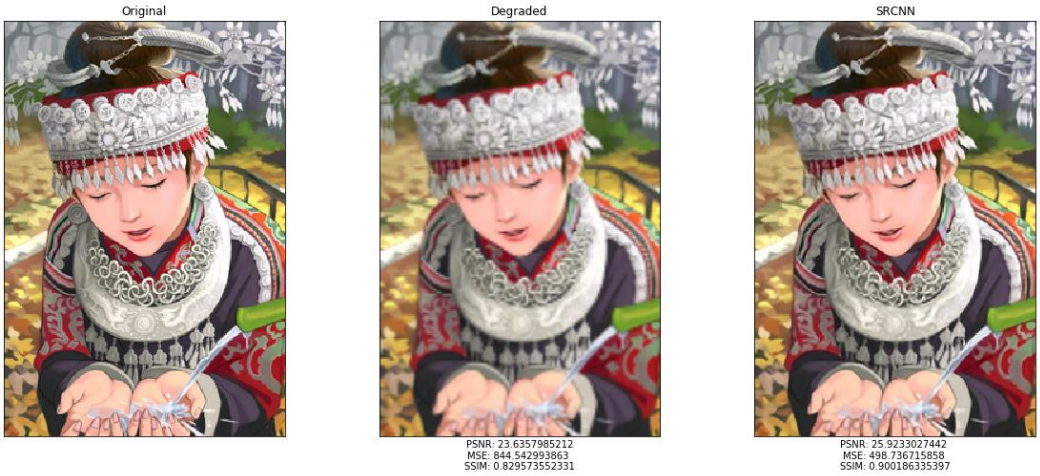
```
    # remove the x and y ticks
    for ax in axs:
        ax.set_xticks([])
        ax.set_yticks([])

    print('Saving {}'.format(file))
    fig.savefig('output/{}.png'.format(os.path.splitext(file)[0]))
    plt.close()
```

```
Saving baboon.bmp
Saving baby_GT.bmp
Saving barbara.bmp
Saving bird_GT.bmp
Saving butterfly_GT.bmp
Saving coastguard.bmp
Saving comic.bmp
Saving face.bmp
Saving flowers.bmp
Saving foreman.bmp
Saving head_GT.bmp
Saving lenna.bmp
Saving monarch.bmp
Saving pepper.bmp
Saving ppt3.bmp
Saving woman_GT.bmp
Saving zebra.bmp
```


Original
Degraded
PSNR: 27.7765644184
MSE: 325.495638621
SSIM: 0.892294789227
SRCNN
PSNR: 30.3501620379
MSE: 179.963906464
SSIM: 0.932882425989


Original
Degraded
PSNR: 31.4472167901
MSE: 139.791233344
SSIM: 0.837672854755
SRCNN
PSNR: 32.7741274104
MSE: 102.988533611
SSIM: 0.849170174733


Original
Degraded
PSNR: 32.0448352334
MSE: 121.819858931
SSIM: 0.938395720061
SRCNN
PSNR: 36.2469889893
MSE: 46.2916559472
SSIM: 0.962863487043


Original
Degraded
PSNR: 23.6357985212
MSE: 844.542993863
SSIM: 0.829573552331
SRCNN
PSNR: 25.9233027442
MSE: 498.736715858
SSIM: 0.900186335397


Original
Degraded
SRCNN

CHAPTER-5
CONCLUSION

We have presented deep learning approach for single image super-resolution (SR). We show that 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. Besides, the proposed structure, with its advantages of simplicity and robustness, could be applied to other low-level vision problems, such as image deblurring or simultaneous SR+denoising. One could also investigate a network to cope with different upscaling factors.

REFERENCES

[1] Aharon, M., Elad, M., Bruckstein, A.: K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. IEEE Transactions on Signal Processing 54(11), 4311–4322 (2006)

[2] Bevilacqua, M., Roumy, A., Guillemot, C., Morel, M.L.A.: Lowcomplexity single-image super-resolution based on nonnegative neighbor embedding. In: British Machine Vision Conference (2012)

[3] Burger, H.C., Schuler, C.J., Harmeling, S.: Image denoising: Can plain neural networks compete with BM3D? In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 2392–2399 (2012)

[4] Chang, H., Yeung, D.Y., Xiong, Y.: Super-resolution through neighbor embedding. In: IEEE Conference on Computer Vision and Pattern Recognition (2004)

[5] Cui, Z., Chang, H., Shan, S., Zhong, B., Chen, X.: Deep network cascade for image super-resolution. In: European Conference on Computer Vision, pp. 49–64 (2014)

[6] Dai, D., Timofte, R., Van Gool, L.: Jointly optimized regressors for image super-resolution. In: Eurographics. vol. 7, p. 8 (2015)

[7] Dai, S., Han, M., Xu, W., Wu, Y., Gong, Y., Katsaggelos, A.K.: Softcuts: a soft edge smoothness prior for color image superresolution. IEEE Transactions on Image Processing 18(5), 969–981 (2009)

[8] Damera-Venkata, N., Kite, T.D., Geisler, W.S., Evans, B.L., Bovik, A.C.: Image quality assessment based on a degradation model. IEEE Transactions on Image Processing 9(4), 636–650 (2000)

[9] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255 (2009)

[10] Denton, E., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems (2014)

[11] Dong, C., Loy, C.C., He, K., Tang, X.: Learning a deep convolutional network for image super-resolution. In: European Conference on Computer Vision, pp. 184–199 (2014)

[12] Eigen, D., Krishnan, D., Fergus, R.: Restoring an image taken through a window covered with dirt or rain. In: IEEE International Conference on Computer Vision. pp. 633–640 (2013)

[13] Freedman, G., Fattal, R.: Image and video upscaling from local self-examples. ACM Transactions on Graphics 30(2), 12 (2011)

[14] Freeman, W.T., Jones, T.R., Pasztor, E.C.: Example-based superresolution. Computer Graphics and Applications 22(2), 56–65 (2002)

[15] Freeman, W.T., Pasztor, E.C., Carmichael, O.T.: Learning lowlevel vision. International Journal of Computer Vision 40(1), 25–47 (2000)