Using The Super Resolution Convolutional Neural Network for Image Restoration

As the title suggests, the SRCNN is a deep convolutional neural network that learns end-to-end mapping of low resolution to high resolution images. As a result, we can use it to improve the image quality of low resolution images. To evaluate the performance of this network, we will be using three image quality metrics: peak signal to noise ratio (PSNR), mean squared error (MSE), and the structural similarity (SSIM) index.

Furthermore, we will be using OpenCV, the Open Source Computer Vision Library. OpenCV was originally developed by Intel and is used for many real-time computer vision applications. In this particular project, we will be using it to pre and post process our images. As you will see later, we will frequently be converting our images back and forth between the RGB, BGR, and YCrCb color spaces. This is necessary because the SRCNN network was trained on the luminance (Y) channel in the YCrCb color space.

During this project, you will learn how to:

1.use the PSNR, MSE, and SSIM image quality metrics,

2.process images using OpenCV,

3.convert between the RGB, BGR, and YCrCb color spaces,

4.build deep neural networks in Keras,

5.deploy and evaluate the SRCNN network

1. Importing Packages

Let's dive right in! In this first cell, we will import the libraries and packages we will be using in this project and print their version numbers.

In [2]:

```
# 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.| (default, May 11 2017, 13:17:2
6) [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
```

2. Image Quality Metrics

To start, lets define a couple of functions that we can use to calculate the PSNR, MSE, and SSIM. The structural similiarity (SSIM) index was imported directly from the scikit-image library; however, we will have to define our own functions for the PSNR and MSE. Furthermore, we will wrap all three of these metrics into a single function that we can call later.

In [5]:

```
# 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 t
wo 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
```

3. Preparing Images

For this project, we will be using the same images that were used in the original SRCNN paper. We can download these images from http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html (http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html). The .zip file identified as the MATLAB code contains the images we want. Copy both the Set5 and Set14 datasets into a new folder called 'source'.

Now that we have some images, we want to produce low resolution versions of these same images. We can accomplish this by resizing the images, both downwards and upwards, using OpeCV. There are several interpolation methods that can be used to resize images; however, we will be using bilinear interpolation.

Once we produce these low resolution images, we can save them in a new folder.

In [6]:

```
# 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)
```

In [7]:

```
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
```

3. Testing Low Resolution Images

To ensure that our image quality metrics are being calculated correctly and that the images were effectively degraded, lets calculate the PSNR, MSE, and SSIM between our reference images and the degraded images that we just prepared.

In [8]:

```
# 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
```

4. Building the SRCNN Model

Now that we have our low resolution images and all three image quality metrics functioning properly, we can start building the SRCNN. In Keras, it's as simple as adding layers one after the other. The achitecture and hyper parameters of the SRCNN network can be obtained from the publication referenced above.

In [9]:

```
# 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_unif
orm',
                     activation='relu', padding='valid', use_bias=True, input_shape=(No
ne, None, 1)))
    SRCNN.add(Conv2D(filters=64, kernel_size = (3, 3), kernel_initializer='glorot_unifo
rm',
                     activation='relu', padding='same', use_bias=True))
    SRCNN.add(Conv2D(filters=1, kernel_size = (5, 5), kernel_initializer='glorot_unifor
m',
                     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_err
or'])

    return SRCNN
```

5. Deploying the SRCNN

Now that we have defined our model, we can use it for single-image super-resolution. However, before we do this, we will need to define a couple of image processing functions. Furthermore, it will be necessary to preprocess the images extensively before using them as inputs to the network. This processing will include cropping and color space conversions.

Additionally, to save us the time it takes to train a deep neural network, we will be loading pre-trained weights for the SRCNN. These weights can be found at the following GitHub page: https://github.com/MarkPrecursor/SRCNN-keras (https://github.com/MarkPrecursor/SRCNN-keras)

Once we have tested our network, we can perform single-image super-resolution on all of our input images. Furthermore, after processing, we can calculate the PSNR, MSE, and SSIM on the images that we produce. We can save these images directly or create subplots to conveniently display the original, low resolution, and high resolution images side by side.

In [10]:

```
# 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
```

In [12]:

```
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], score
s[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
```

In [13]:

```
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
```