IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 22, Issue 3, Ser. I (May - June 2020), PP 46-53
www.iosrjournals.org

Perfect Single Image (SR) Super Resolutionwith (SRCNN), Deep SuperResolutionConvolutional Neural NetworkandOpenCV Method.

Hasan Murshed[1],Zhenhua Wei[2], Mona Ahmed[3]
[1, 2,3]*(College of Control and Computer Engineering, North China Electric Power University, R.P.China)*

Abstract:
In this paper we suggest and proposed a new conceptual method layout of image super resolution (SR) restoration with SRCNN and Opencv combination. The objective of (SR) super resolution is to recuperate from a low-resolution input a high resolution image. We will also use the Library of Open Source Computer Vision. Open CV. Originally developed by Intel, Opencv is used for many applications for real time computer vision. (In this specific project, were going to use it to pre-and post our images). We must transform our images between the RGB frequently, BGR and the color spaces of Ycrcb. This is important due to the training of the SRCNN network in the Ycrcb color space on the luminance (Y) channel.

Key Word:*Super-Resolution; Deep Convolutional NeuralNetworks; Machine Learning; Regularization;Patch-Extraction; Non-Linear Mapping; Image reconstruction.*

Date of Submission: 01-05-2020 Date of Acceptance: 14-05-2020

I. Introduction

In this research paper, a (SRCNN) was applied and evaluated using the post processing approach, (which is the use for enhancing image resolution emerging with deep learning super resolution method and SRCNN using the training data collection, because the SRCNN was trained. The proficient SRCNN was used to recreate a high-resolution image of a low resolution image, which was taken from the original image of the test. Two image quality parameters are calculated for quantitative analysis and contrasted with techniques of traditional linear interpolation. The reliability of the SRCNN image restoration process was suggestively advanced than methods of linear interpolation. The high-resolution picture recovered by the SRCNN system has been fully preserved and, in fact, equivalent to the initial reference image for a magnification of roughly x2. These results suggest that the SRCNN system outperforms the linear method to increase image resolution significantly and the results suggest that SRCNN can be used to create standard images to high resolution images.

The SRCNN authors depict their network and display the corresponding sparse-coding method4 of their method, a commonly used program of picture SR learning.

An essential and informative feature of their work because it demonstrates how examples can be applied to popular models of CNN.

The SR super resolution objective is to retrieve a low-resolution input a high resolution image. As indicated by the name, the SRCNN is a profound neural network capable of mapping low resolution to high resolution images from one end to the next. This allows us to use the image of low resolution images to improve their quality. We use three image quality metrics: the maximum noise signal ratio (PSNR); the mean squared error (MSE), and the structural semblance (SSIM) indices to evaluate the performance of this network.
There are the following operations of the SRCNN:

1. Preprocessing; upscale the image is of low resolution to the desired high resolution.
2. Extract function; Extracts from an up scaled LR image a set of feature maps.
3. Nonlinear mapping; Maps of low resolution to high resolution patches feature maps.
4. Reconstruction; Proceeds from high resolution image from high resolution patches.

As we implies the title, the SRCNN is an advanced neural network that knows how to model low resolution objects at high-resolution end-to-end. It helps us to use it to enhance the image quality of pictures that are of low resolution. To measure this network's efficiency, we will use three performance indicators:

PSNR (peak signal to noise ratio), MSE (mean squared error) and SSIM (Structural Similarity Index). In contrast, we will use three quality metrics. In short, we can get better quality of a larger image with a better SR approach even if we get a little image at first [1][2].

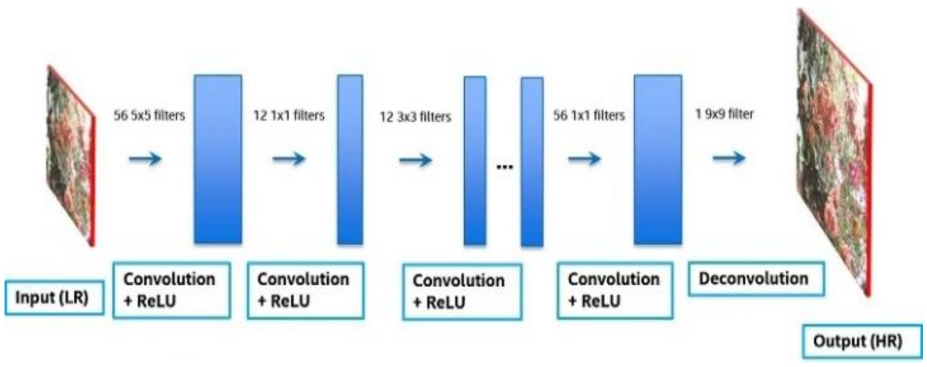
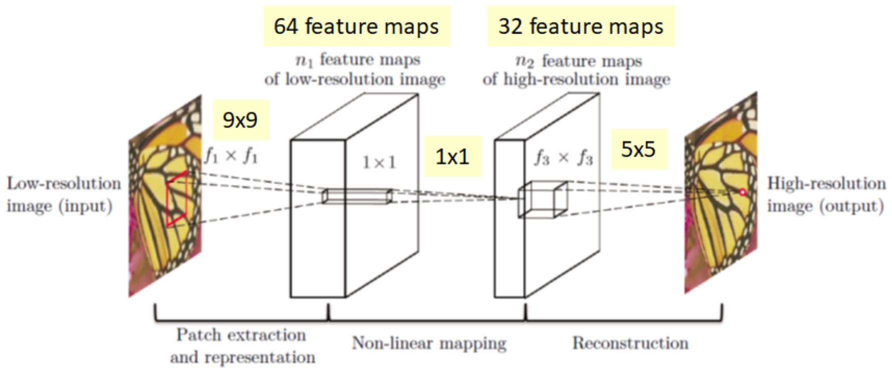


Fig 1. Degraded to reconstructed image

II. Network For SRCNN



64 feature maps
n_1 feature maps of low-resolution image
32 feature maps
n_2 feature maps of high-resolution image
9x9
$f_1 \times f_1$
1×1
1x1
$f_3 \times f_3$
5x5
Low-resolution image (input)
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Fig 2. Restoration of better quality of a larger image with a better SR approach (Image source SRCNN by Chao Dong et al)

III. Collection Of Libraries And Packages

The Initially, we will import and print out version numbers of the libraries and packages we will use in this project. This is a significant step in ensuring that everyone are on the same page.

Allowing others to replicate the results we get.[2]

```
#check package versions
import sys
importkeras
import cv2
importnumpy
importmatplotlib
importskimage
print (“Python: {}”. Format(sys. version))
print(Keras: ['format(keras._version_J)
Import the necessary packages
fromkeras.models import Sequential
fromkeras.layers import Conv2D
fromkeras.optimizers import Adam
fromskimage.measure import compare_ssim as ssim
frommatplotlib import pyplot as plt

import cv2
importnumpy as np
import math
importos
# python magic function, displays pyplot figures in the        notebook
%matplotlib inline
```

IV. Metrics Of Image Quality

Let's describe a few functions to measure SSIM. PSNR and MSE in order to start with, we imported the Structural Similarity (SSIM) index directly from the Science Kit Image library but we will have to define our own PSNR and MSE functions. In addition, these three metrics are wrapped in a single function we will manage further.

```
# define a function for peak signal-to-noise ratio (PSNR)
defpsnr(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)
defmse(target, ref):
 # the MSE between the two images is the sum of the squared difference
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
defcompare_images(target, ref):
scores = []
scores.append(psnr(target, ref))
scores.append(mse(target, ref))
scores.append(ssim(target, ref, multichannel =True))
return scores
```

V. Image Restoration And Preparation With SRCNN

Use Keras to deploy (SRCNN)

The SRCNN is a deeply convolutionary network of neural that learn to end-to-end low resolution images to high resolution. In this method that the quality of images of low resolution images can be improved. Use image quality measurements to measure the performance of this network with:[3][5]

- PSNR, (peak signal to noise ratio.
- MSE, (mean squared error).
- SSIM, (structural similarity index)
- Using Opencv for process images.
- Converting images among the Ycrcb color spaces, RGB and BGR.
- Prepare image with SRCNN network in Keras.
- SRCNN network, evaluate and deploying

We will also, pre-process and post our pictures using OpenCV.

Our images are also converted back and forth frequently between the color spaces RGB, BGR and Ycrcb. The SRCNN network was trained in the color area of the Ycrcb on the luminance(Y) channel.[3]

VI. Method

First we preparing degraded pictures by introducing excellence alterations by resize rise and depressed new degraded images are of equal resolution to the images of base. By resizing the image, we save the original pixel information in minor areas so that the information was lost when the image was sized. This has been accomplished using Opencv"cv2 library" as it is designed for a computer vision application in real time. The despoiled images are placed in the foundation directory [4].

SRCNN Model Construction:

```
# 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='rel
u', 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='linea
r',padding='vali d', 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
```

VII. Image Processing

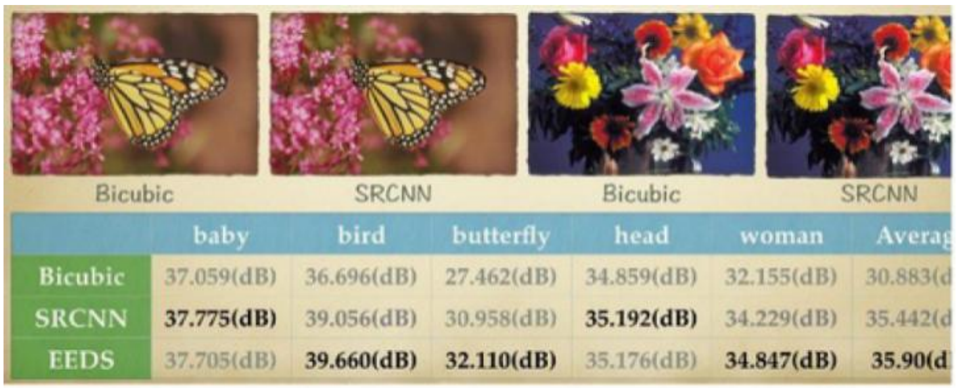
This will include processing conversions for cropping and color pace weight the reference and degraded images: in opencv, images will be loaded as channels of BGR modcrop(BGR): # necessary because when we run images through SRCNN based on kernel sizes and convolutional layers, some of these external pixels will be lost, the images will become smaller and that's why it is.Pre-processing the images: This processing will include cropping and color space conversions, load the degraded and reference images, in opencv, images are loaded as BGR channels [5][6].Modcrop(): #necessary because when we run images through SRCNN based on the kernel sizes and convolutional lavers, we are going to lose some of these outside pixels, the images are going to get smaller and that' s why it is necessary to have a divisible image size,ie, divisible by scale by cropping the Images size.shave(): crop offs the border size from all sides of the image load(deploy)the SRCNN: To save us the time it takes to train a deep neural network, we will be loading pre-trained weights for the SRCNN.midcrop(): # required because if we run SRCNN images based on kernel sizes and convolutional lavers Well lose some of these visible pixels the images will be smaller and that is why it is important to have a divisible image size, that is, divisible by scale by the width of the image.

Shave(): crop offsets the image boundaries on all sides SRCNN load deploy: Save us time to train a deep neural network we're going to load pre-trained SRCNN weights

My execution differs from the original paper, including:

- Using Adam algorithm with a learning rate of 0.0003 for all lavers for optimization.
- The openev library, not the matlab library, should be used to produce the training data and the tested data.
- I didn't set different learning rates in different layers, but it still worked.

Ycrcb has also some variations in colors on Matlab and Opencv. So you can use the code written with matlab if you wish to compare your results with research articles.


Bicubic
SRCNN
Bicubic
SRCNN

	baby	bird	butterfly	head	woman	Averag
Bicubic	37.059(dB)	36.696(dB)	27.462(dB)	34.859(dB)	32.155(dB)	30.883(d
SRCNN	**37.775(dB)**	39.056(dB)	30.958(dB)	**35.192(dB)**	34.229(dB)	35.442(d
EEDS	37.705(dB)	**39.660(dB)**	**32.110(dB)**	35.176(dB)	**34.847(dB)**	**35.90(d**

Fig 3. Result training for epochs on images with upscaling factor

Model testing: Once we've checked our network on all our input images, we can perform a single image super resolution. In addition, we can measure the MSE, SSIM and PSNR, on the Images we generate after processing. Convert the Ycrcb image "image 3 channel-Y channel srcnn trained". Produce object standardize and slice because SRCNN works on 3D depth inputs I one-channel inputs or one-dimensional inputs.

Implement the srcnn super resolution:

The prediction image back to its range after the system output normalized (0-255)

The prediction channel, Socopy Y, canal is back to image and converted to BGR only exists in the prediction image.Remove the border from the reference and degraded image so that all our images are of the same size (ref degraded, (low, res), and output, (high, res)) [7][8].

VIII. Calculation Of Image Quality Metrics

All metrics of image quality have improved. PSNR, improved MSE, destroyed and SSIM, increased from deteriomted image to restored image save to the output directory new high-resolution images, and save the reconstructed, high resolution images to folder. [8]

Other than books, capitalize only the first word in a paper title, except for proper nouns and element symbols. For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation See the end of this document for formats and examples of common references. For a complete discussion of references and their formats, see the IEEE style manual at www.ieee.org/authortools.

We now have some image, we want to create versions of these same images in low resolution. We can do this by redimensioning the images, Use Opecv both downwards and upwards. There are many methods of interpolation for resizing images; we' Il use bilinear interpolation, however.

Formerly we have produced these images with low resolution we can save it in a new directory [9].

```
# prepare degraded images by introducing quality distortions via resizing
defprepare_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)
```

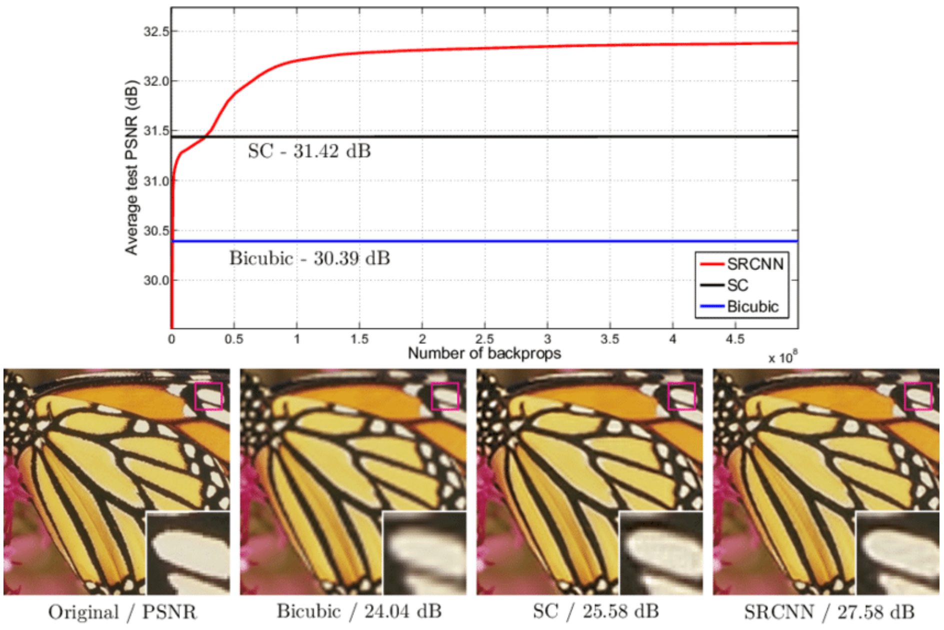


Fig 4. With a few training iterations, the proposed SRCNN exceeds the bicubic baseline, and performs with moderate training the SC, sparse coding based method. With more training iterations, the performance can be further improved.

IX. Check Images With Low Resolution

To make sure image quality measurements are accurately measured and images are degraded, let’s compare between our reference images. PSNR. MSE and SSIM and the degraded images we've just produced. To ensure that our image quality metrics are being calculated correctly and that the images were effectively degraded, lets calculate the SSIM PSNR and MSE and between our reference images and the degraded images that we just prepared.

```
# 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]))
```

When our network has been checked, on all our input images we can perform a single image super resolution. In addition, we can measure the SSIM. PSNR and MSE and on the images we generate after processing. These images can be saved directly or sub-plots created to view side by side the actual low resolution and high resolution images. [10]

```
# define necessary image processing functions
defmodcrop(img, scale):
tmpsz = img.shape
sz = tmpsz[0:2]
sz = sz - np.mod(sz, scale)
img = img[0:sz[0], 1:sz[1]]
returnimg
def shave(image, border):
img = image[border: -border, border: -border]
returnimg# 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, scoresref, 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
```

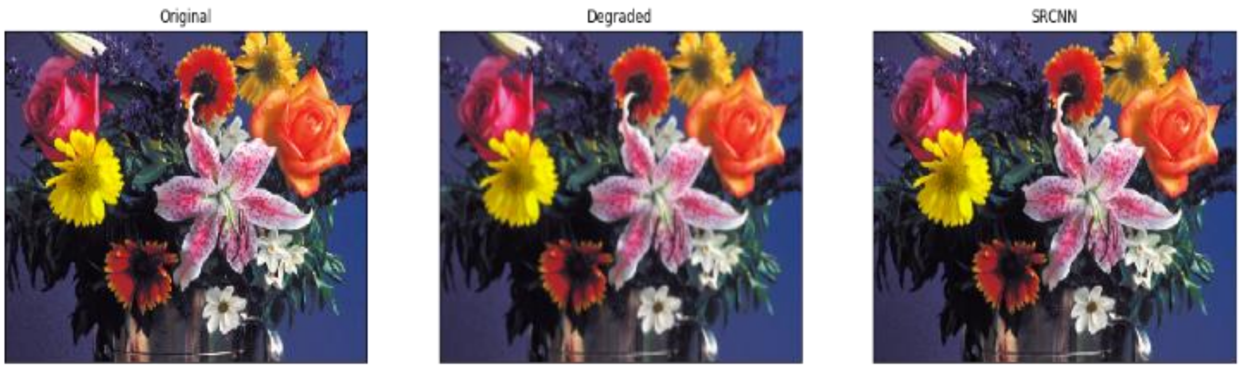


Fig5. Output result of LR to HR image

```
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()
```

X. Conclusion

Microgrid becomes an important aspect of the smart grid of the future, featuring great flexibility in operation, improved quality and enhanced reliability of power. Network implementation and energy management techniques are important aspects of the microgrid, this helps the microgrid to work well in both and stand alone and grid connected mode. This research provides an overview of technologies for micro 1 id energy management and grid integration strategies.

This shows that the recent trend in research on the DG interface converter focuses on small size, improved efficiency, modular sign and multi-port design. A hybrid mix of contact-free and communication energy management technologies could be a good balance of process optimum operation, resilience and efficiency for the energy management strategy. This paper also the interfacing converter control schemes and shows that VCM-based methods are gaining more attention because of their ability to mimic a synchronous generators behavior. This paper fully addresses DGs ancillary services. The auxiliary facility develops a talented topic for advance supporting grid control improving the feature of grid power and at the same time improving the cost effectiveness of DGs and microgrids based on electronic power.

References

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

[2] T Bluche: transcription and segmentation of Joint line for end to end handwritten paragraph recognition. (2016)

[3] A. Roumy, and M. L. and M. Bevilacqua: Morel Lowcomplexity single image to super resolution on nonnegative neighbor embedding. 2012

[4] Loy, C.C., and Dong, C: SRCNN, Learning a deep convolutional network for image super resolution (2014).

[5] D-Y. Yeung. and H. Chang:, Super resolution through neighbor embedding, 2004.

[6] C Dong. C. C. Loy, K. He, and X. Tang Learning a deep convolutional network for image super-resolution In ECCV pages184-199,2014. 2,7

[7] X. Tang and C Dong. C. IEEE Transactions on Machine Intelligence, And Pattern Analysis 2016.

[8] T. S. Huang. and J. Yang, IEEE Transactions on Image Processing 2017.

[9] T. R. Jones, and W. T. Freeman: Example based super resolution /EEE Applications and Computer Graphics 2002.

Hasan Murshed, et. al. "Perfect Single Image (SR) Super Resolution with (SRCNN), Deep Super Resolution Convolutional Neural Network and Open CV Method." *IOSR Journal of Computer Engineering (IOSR-JCE)*, 22(3), (2020), pp. 46-53.