Single Image Super Resolution

Guntaka Abhishek - 0891103

Abstract - I have used a deep learning super-resolution model which is an end to end model that will map the low resolution and the high resolution. We take the low-resolution image as the input and we generate a high-resolution as the output. The algorithm goes in the following order : Preprocessing, Feature extraction, Non-linear mapping, and Reconstruction. They will be further explained in the report.

1.Introduction

Super-resolution has been a computer vision problem for a long period of time and it is still not reached its pinnacle. The method that has been used in the project is one of the methods that use internal similarities of the same image.

This is achieved from the pipeline of the model, starting with preprocessing the input image (i.e downscaling and upscaling the image by a factor) then the preprocessed image will look blurry and loses a lot of detail in the image, the current image is our low-resolution image and the image before preprocessing will be our reference high-resolution image and then the reconstruction of patches of the high-resolution output begins to produce the final output.

The used model is called Super-Resolution Convolution Neural Network(SRCNN)[1], The used model has various robust properties. First, the structure is simple as well as the accuracy is high compared to a few other methods. Second, with a moderate number of filters and layers, this method achieves fast speed for practical online usage even on a CPU. This method is faster than a number of other example-based methods.

2. Operations Performed

There are four key operations that have to be performed:

1) Patch Extraction
2) Non-linear Mapping
3) Reconstruction

Patch Extraction -

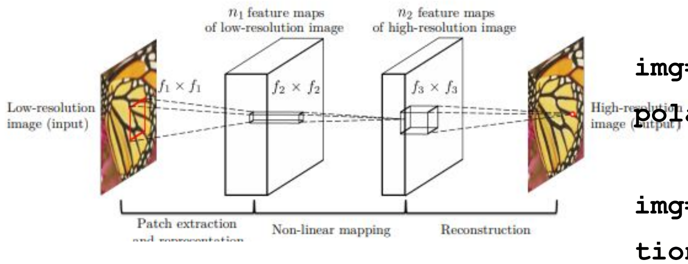
This operation extracts patches from the Low-resolution image and represents each patch as a vector (High-dimensional). These vectors contain a set of feature maps

Non-linear mapping -

This operation non linearly maps each high-dimensional vector. Each mapped vector is a high-resolution patch replica.

Reconstruction -

This operation sums the above high-quality resolution patch wise representations to create the final image.



Algorithm:

Step1 - Take the ground truth image **x** and Downsize it and then upscale the downsized image (downsize it by a factor - ex 2,4 and etc) to obtain the low-resolution image **y**

Step2 - The obtained image **y** will now go through a series of trimming and cropping operations

Step 3 - After these operations, the image is sent into the custom model (sequential model with additional layers added to it) for non-linear mapping of features of the image and the ground truth image features to generate High-resolution patches.

Step 4 - These patches are merged to make one whole image and the image generated is the final output.

3. Pseudocode -

Here is the code for all the steps in the algorithm:

Step 1:

```
def image_pre(file,factor):
  img=cv2.imread(file)
  h,w,_ =img.shape
  n_h=int(h/factor)
  n_w=int(w/factor)

img=cv2.resize(img,(n_h,n_w),inter
polation = cv2.INTER_LINEAR)

img=cv2.resize(img,(w,h),interpola
tion=cv2.INTER_LINEAR)

cv2.imwrite('processed_img.jpg',im
g)
```

Step 2 :

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

Step 3 :

```
def model():
     custom_model = Sequential()

custom_model.add(Conv2D(filters=12
8, kernel_size = (9, 9),
kernel_initializer='glorot_uniform
',

activation='relu',
```

```
padding='valid', use_bias=True,
input_shape=(None, None, 1)))

custom_model.add(Conv2D(filters=64
, kernel_size = (3, 3),
kernel_initializer='glorot_uniform
',

activation='relu', padding='same',
use_bias=True))

custom_model.add(Conv2D(filters=1,
kernel_size = (5, 5),
kernel_initializer='glorot_uniform
',

activation='linear',
padding='valid', use_bias=True))
    adam = Adam(lr=0.0003)

custom_model.compile(optimizer=ada
m, loss='mean_squared_error',
metrics=['mean_squared_error'])
    return custom_model
```

Step 4 :

```
def predict(image_path):
    custom_model = model()

custom_model.load_weights('/conten
t/custom_weights.h5')
    proc_img =
cv2.imread(image_path)
    ref =
cv2.imread('/content/GOKU.bmp')
    ref = modcrop(ref, 3)
    proc_img = modcrop(proc_img,
3)
    temp = cv2.cvtColor(proc_img,
cv2.COLOR_BGR2YCrCb)
    Y = np.zeros((1,
temp.shape[0], temp.shape[1], 1),
dtype=float)
    Y[0, :, :, 0] = temp[:, :,
0].astype(float) / 255
    super =
custom_model.predict(Y,
batch_size=1)
    super *= 255
    super[super[:] > 255] = 255
    super[super[:] < 0] = 0
    super = super.astype(np.uint8)
    temp = trim(temp, 6)
    temp[:, :, 0] = super[0, :, :,
0]
    output = cv2.cvtColor(temp,
cv2.COLOR_YCrCb2BGR)
    ref =
trim(ref.astype(np.uint8), 6)
    proc_img =
trim(proc_img.astype(np.uint8), 6)
    return ref, proc_img, output
```

4. Output

Here are the results -

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