Deep Learning for Super Resolution and Applications

Zahraa Hasan

University Of Applied Science, Faculty of Literature and Science, Manama, Bahrain
Email: Zahraamathscience1243@gmail.com

Abstract

High-resolution technologies are aimed at obtaining a high-resolution image from a low-resolution image, and the importance of this field has increased due to the emergence of the need to have high-resolution images in many important applications such as medical, security, and other images. Methods for obtaining ultra-high-resolution images have developed after the advent of Deep Learning Technologies, which have shown good results in this task, Due to the importance of the field of ultra-high-resolution images and deep learning, In this article we will explain one of the deep learning models used to obtain a high-resolution image from a low-resolution image and how to build and train it based on one of the famous deep learning offices and using one of the google platforms used in training, namely Google Laboratory

Keywords: Deep learning; high-resolution; images; low-resolution

1. Introduction:

High-resolution images are images that have a larger number of pixels per inch, that is, they have a larger amount of information (a larger amount of information means more details in

The image is like thin lines and protruding edges... Etc.) [1]

The super-resolution process is defined as the process of retrieving a high-resolution image from a low-resolution image [5]

These images are of great importance due to the need for them in many uses such as medical applications, smart surveillance, remote sensing remote sensing and computer vision tasks computer vision tasks [2]

It is not necessarily possible to obtain high-resolution images despite the development of digital imaging devices due to external influences that can cause image distortion or reduced details in the image when taken, for example, when taking pictures of someone's face located at a far distance from the camera, the details of the face are blurred or the landmarks features of this person's face cannot be obtained and will therefore affect the possibility of identification. It is also not possible to use low-resolution images taken with unsophisticated photographic devices [2], therefore the process of obtaining high-resolution images from low-resolution images will inevitably be important. The images resulting from the process of converting low-resolution images to high-resolution are known as super-resolution images.

2. Methods for obtaining ultra-high resolution images

IMAGE SUPER-RESOLUTION TECHNIQUES

Ultra-high-resolution images can be obtained by applying image processing techniques such as the Frequency Domain Approach, Spatial Domain Super-Resolution Methods, and others [3], but here we will discuss methods based on deep learning in obtaining the desired images.

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

Obtaining high-resolution images using deep learning models A.

Deep learning models have shown positive results in obtaining high-resolution images from low-resolution images, as models have evolved from serial models such as SRCNN, and even current models based on generative adversarial networks have given results even in models with few bypass layers, The SRCNN model has only 3 neural networks and shows good results in

Therefore, interest in the development of such models has increased [5-6].

B. The evolution of deep learning models used to obtain high-resolution images

Initially, sequential models trained on pairs of high-resolution and low-resolution images were used as supervised models, that is, the data set consists of low-resolution images and corresponding high-resolution images, therefore, when training the model, the low-resolution image will be used as input, and then the parameters will be updated depending on the calculation of the error between the images generated by the model (model output) and the existing high-resolution images.

In this article, a detailed explanation will be given about the Super Resolution Convolution Neural Networks SRCNN model used in the process of extracting high-resolution images from low-resolution images [4]

The structure of the model:

It is considered one of the simplest deep learning models used to obtain ultra-high-resolution images, which consists of three convolutional layers possessing the following parameters:

Table 1: Components of the SRCNN model

Class	Number of candidates	Filter size	Continue activation
The first	128	9*9	rel
The second	64	3*3	relu
The third	1	5*5	linear

The practicality of using the model:

The model can be built depending on one of the popular frameworks such as Tensorflow, Keras, or Pytorch, here the Keras of Tensor flow will be used for ease of Use and understanding for any beginner.

The Google Colab work environment provided by Google can be used to apply the model in practice and is similar to Jupiter Notebooks which offers free GPU and Hard Disk storage space up to GB 107.72 and RAM up to GB 12.69 currently.

∗**Creating a working environment:**

This includes creating a notebook and saving it directly to the developer's Google Drive, and to do this, this is done by pressing the right button, then selecting more from the drop-down menu, and then choosing the Colaboratory option, i.e.:

Right Click→More→Colaboratory

The following figure shows an illustration of the previous step:

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

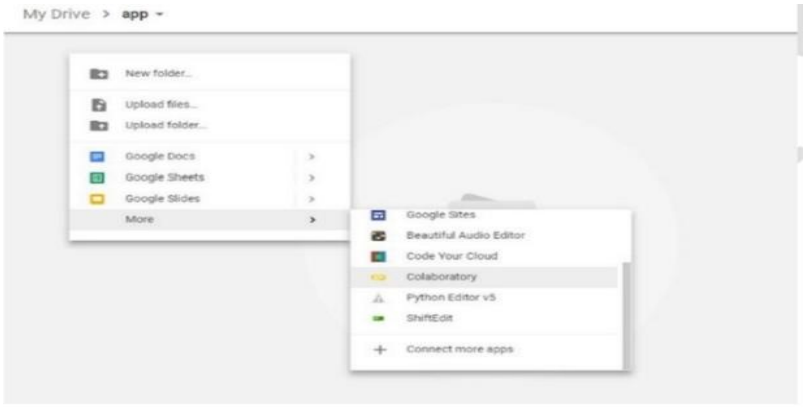

Figure 1: Create a notebook and save it directly to Google Drive

:The notebook will appear as Unititled0 as in the following figure

Figure 2: The new notebook window .

Of course, the name can be changed by typing the new name in place of the current one, and we will change the name to Tutorial.

This notebook uses the CPU to train by default and to change it to GPU or TPU you must click on the runtime option and then choose change runtime type The following figure appears

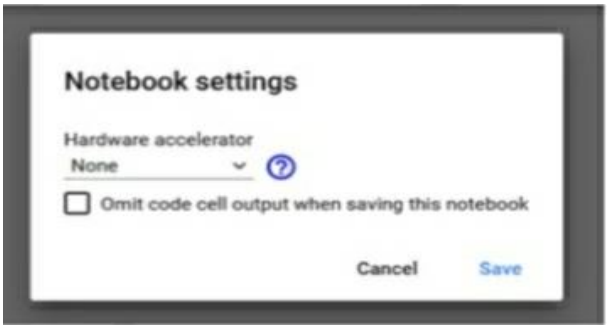

Figure 3: The window for changing the type of steel gear used in training

We click on the arrow next to None and then we select the GPU as it appears:

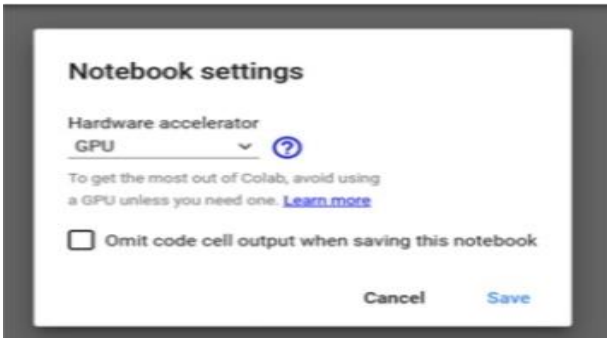

Figure 4: Choosing the GPU for training

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

Then we press the save button and the work environment is ready.

∗**Include libraries:**

Python language instructions can be written in a cell and then executed by clicking the Run button at the beginning of each cell or pressing shift and then enter, which executes and moves to a new cell inside the cell we will call the necessary libraries to build the form, as shown in the following code

Figure 5: Calling the necessary libraries for model building and training

Explanation of instructions:

The first line: is to call the tensorflow library and declare it as tf so that tf is used instead of tensorflow in subsequent instructions.

The second line: using the Sequential form from the TensorFlow library.Keras and the models class (it should be noted that in the serial model, layers are added sequentially and can be added branchwise using a model called Model)

The third line: call the classes building classes from the TensorFlow library. Keras and the layers class and these classes are Input to form the income layer which is the first layer in the model and Conv2D to form the bypass layers of the model.

The fourth line: calls for the optimization follower Adam and the optimization followers are algorithms used to change the qualities of the model such as weights and learning coefficient to reduce the loss There are many types, the most famous of which are gradient descent, SGD, Adam, and others.

The fifth line: using the Model Checkpoint to save weights and biases.

Sixth line: Call the open cv Library, which is a well-known library used to perform operations on images and includes a lot of important dependencies in image processing.

Seventh line: Call NUMBY library as NP which is a famous library dealing with arrays.

The eighth line: call matplotlib.PYPLOT for displaying images.

Ninth line: Call the math library for calculations.

The Tenth Line: Call the OS library to deal with files and folders of creation, deletion, and others.

Eleventh line: Call the H5 file handling library

Formation of the income data set of the model:*

2002x 200 1 100 x 100 1 100 x 100 x 200 x 2 2.

SR SRKN takes an input equal to the output size the image is re-enlarged to the basic dimensions, in other words:

The first step:

LR display = hour/scale display.

LR height = hour height/scale.

Doi: https://doi.org/10.54216/GJMSA.080204

Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

The second step:

Final offer=HR offer.

High_final = hour high.

The following figure shows the necessary instructions for obtaining the low-resolution image:

```
#two resize operation to produce training data and labels
lr_img2 = cv2.resize(hr_img, (shape[1] // 2, shape[0] // 2))
lr_img2 = cv2.resize(lr_img2, (shape[1], shape[0]))
```

Figure 6: The code for obtaining low-resolution images

Model construction:

The model consists of three layers and with specific coefficients as mentioned earlier therefore to build the model we will write:

```
#building the model
def SRCNN():
    SRCNN = Sequential()
    SRCNN.add(Conv2D(128,(9,9),activation='relu', padding='valid',use_bias=True,
                     input_shape=(32, 32, 1)))
    SRCNN.add(Conv2D(64,(3,3),activation='relu', padding='same', use_bias=True))
    SRCNN.add(Conv2D(1,(5,5),activation='linear', padding='valid', use_bias=True))
    adam = Adam(learning_rate=0.0001)
    SRCNN.compile(optimizer=adam, loss='mean_squared_error', metrics=['mean_squared_error'])
    return SRCNN
```

Figure 7: Model building code

We define the SRCNN model as a serial model in the first line, after which the first layer is added in the next line with the following parameters:

filters with dimensions 9x9 for each filter and a RELU activation follower(an 128 activation follower is a mathematical follower that changes the value of the weights in one layer before passing them to the next layers and a RELU follower, allowing positive values

By moving to the next layer as it is, negative values make it zero). Without a padding, This is indicated in the phrase:

Padding=’valid’

Padding is a technique to make the image size appropriate for the operations to be applied as the convolution layer changes the dimensions of the input image.

When using a valid type padding, the size of the income matrix is kept the same, If the income matrix with dimensions (h X w) becomes (h_{new}X w_{new}), the dimensions resulting from passing the income on the convolution layer according to a mathematical relationship are:

$$dimension = \frac{dimension - filter_{size} + 2 * padding}{stride} + 1$$

Where:

dimension: One of the dimensions of the image is either length or width.

Filter size: is the size of the filter used and in this layer is 3.

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

padding: the size of the income is maintained the same even after passing through the convolution layer, if it is of the same type, by increasing the number of zero lines and columns on the income Matrix, but if it is valid, it allows changing the dimensions of the income.

stride: determines the number of lines and columns that will be crossed when the bypass operation is performed by the filter and in our example is always 1 (default value)

When you need to use the bias, this is done by giving the use _bias parameter the value True

The dimensions of the income are determined using the input_shape parameter and here we make the dimensions (32,32,1), that is, the income should have a height of 32 and a width of 32, but with one channel, any image is gray. The same is true for the rest of the layers, with the difference in terms of the number of filters, their dimensions, and the type of padding, noting that the activation follower in the last layer is linear, that is, it allows the value to pass as it is, and then the definition of the optimization follower adam, as follows:

```
adam = Adam(learning_rate=0.0001)
```

The learning rate expresses the value that controls the amount of change in weights and biases in the network during the learning process so that if the learning rate is large the accuracy of the model will decrease but the training will happen very quickly, while if the learning rate is small the accuracy of the model will increase but the training will happen slowly and therefore choosing an appropriate value for the learning rate is important.

***Translation of the model compiling the model:**

That is, make this model with a suitable physical presence in memory, and it appears in the following code:

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

The translation process needs to determine the optimization dependent, and here The Adam dependent was used, and determine the loss dependent (which is The Dependent that calculates the difference between the output of the actual model and the real output required to be accessed), which is here mean_squared_error, and this type of loss dependent calculates the difference between each pixel of the output image with the corresponding desired real output, squaring this difference, and then calculating the total sum of the square of the differences. As for the metrics parameter, it is optional and it is to determine which value we want to display the values which changed during the training process and here we want to focus on the error values.

Model training:*

It is the final stage where the model is trained on Gray input images using the fit dependent that appears in the following code:

```
srcnn_model.fit(data, label, batch_size=128,
                validation_data=(val_data, val_label),
                callbacks=callbacks_list,
                shuffle=True, epochs=100, verbose=1)
```

The parameters used within the previous code will be explained as follows:

Data: is the Income Data (low-resolution images)

label: real output data approved for input (high-resolution images)

batch_size the number of income items that will be trained on each iteration.

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

validation_data: validation elements are a pair of low-resolution and high-resolution images to test the model on untrained elements to increase the accuracy of the model and check for overfitting(unwanted value)

Callbacks: to execute specific instructions after completion of each iteration.

shuffle: to mix the training data before each iteration to increase the accuracy of the model.

epochs: to determine the number of training times

verbose: to determine the form of training

The code that loads the training, verification, and checkpoint data to maintain the weight values and biases during the training process:

```
#training the model
def train():
    srcnn_model = SRCNN()
    data, label = read_training_data("./crop_train.h5")
    val_data, val_label = read_training_data("./test.h5")

    checkpoint = ModelCheckpoint("SRCNN.h5", monitor='val_loss', verbose=1, save_best_only=True,
                                 save_weights_only=False, mode='min')
    callbacks_list = [checkpoint]
    #start the training
    srcnn_model.fit(data, label, batch_size=128, validation_data=(val_data, val_label),
                    callbacks=callbacks_list, shuffle=True, epochs=100, verbose=1)


if __name__ == "__main__":
    train()
```

|Testing the model on untrained income images :*

Initially, the structure of the form will be loaded as follows:

srcnn_model=SRCNN()

The second step is to load the weights resulting from the training by the line:

scrnn_model.load_weights(model_pth(

Where model_pth is the path of the model weights generated by the training process. Formation of a test image for the compatibility of the model's income. The following code shows the process of forming the approval test image.

```
Y_img = cv2.resize(img[:, :, 0], (shape[1] // scale, shape[0] // scale), cv2.INTER_CUBIC)
Y_img = cv2.resize(Y_img, (shape[1], shape[0]), cv2.INTER_CUBIC)
img[:, :, 0] = Y_img
img = cv2.cvtColor(img, cv2.COLOR_YCrCb2BGR)
#save the input (low resolution) image
cv2.imwrite(INPUT_NAME, img)

Y = np.zeros((1, img.shape[0], img.shape[1], 1), dtype=float)
Y[0, :, :, 0] = Y_img.astype(float) / 255.
```

Doi: https://doi.org/10.54216/GJMSA.080204
Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

To get the output of the model, the following code line can be used:

```
pre = srcnn_model.predict(Y, batch_size=1) * 255.
```

The following code shows the process of storing the output of the model, which is the Super-Resolution image:

```
pre = srcnn_model.predict(Y, batch_size=1) * 255.
pre[pre[:] > 255] = 255
pre[pre[:] < 0] = 0
pre = pre.astype(np.uint8)
img = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
img[6: -6, 6: -6, 0] = pre[0, :, :, 0]
img = cv2.cvtColor(img, cv2.COLOR_YCrCb2BGR)
#save the output (super resolution) image of the model
cv2.imwrite(OUTPUT_NAME, img)
```

To calculate the PSNR which expresses the amount of difference in the pixels of two images (the output of the model is the Super-Resolution image and the actual image, i.e. with high resolution), we write the following lines of code

```
# psnr calculation:
im1 = cv2.imread(IMG_NAME, cv2.IMREAD_COLOR)
im1 = cv2.cvtColor(im1, cv2.COLOR_BGR2YCrCb)[6: -6, 6: -6, 0]
im2 = cv2.imread(INPUT_NAME, cv2.IMREAD_COLOR)
im2 = cv2.cvtColor(im2, cv2.COLOR_BGR2YCrCb)[6: -6, 6: -6, 0]
im3 = cv2.imread(OUTPUT_NAME, cv2.IMREAD_COLOR)
im3 = cv2.cvtColor(im3, cv2.COLOR_BGR2YCrCb)[6: -6, 6: -6, 0]

print("bicubic:")
print(cv2.PSNR(im1, im2))
print("SRCNN:")
print(cv2.PSNR(im1, im3))
```

Results

The test was carried out on the image of the famous butterfly from the datasetset 5. Figure (8) shows the three formulas of the tested image:

The first image from the left is the High-Resolution image•

The image in the middle is the low-resolution image•

The rightmost image is the Super Resolution image•

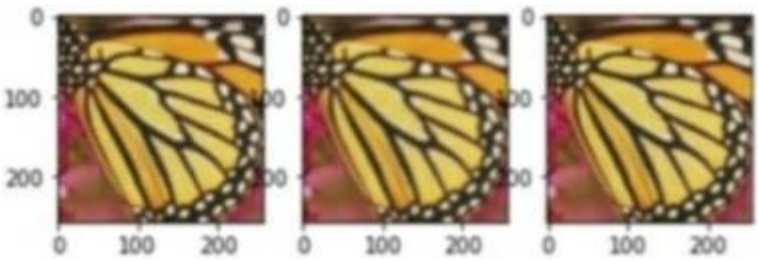

Figure 8: Different versions of the image

And the PSNR values are:

BICUBIC: 24.69

SRCNN: 30.35

Doi: https://doi.org/10.54216/GJMSA.080204

Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023

Where BICUBIC: is the image that has been enlarged and corrected using the interpolation called BICUBIC: which is one of the approximation techniques used in image processing.

3. Conclusion

In this article, we have explained the importance of having super-resolution images and one of the deep learning models used to obtain super-resolution images and the practical application of this model by testing it on an image from a famous data set This article is a start for anyone who wants to enter the field of deep learning and super-resolution, which is one of the fields of image processing.

The full code is available in the following repository;

https://github.com/ReemJbily/SRCNN-googlecolaboratory-tutorial

For expansion, you can return to the original repository at the following link:

https://github.com/MarkPrecursor/SRCNN-keras

References

[1] Gaidhani, P. (n.d.). *Super Resolution. 2021*

[2] Chen, H., He, X., Qing, L., Wu, Y., Ren, C., & Zhu, C. (2021, Mar 3). Real-World Single Image Super-Resolution. p. 18.

[3] Karimi, E., Kangarloo, K., & javadi, S. (2014, Feb). A Survey on Super-Resolution Methods for Image Reconstruction.

[4] Dong, C., Loy, C., He, K., & Tang, X. (2015, Jul 31). Image Super-Resolution Using Deep. p. 14.

[5] Wang, Z., Chen, J., Hoi, S., Fellow, & IEEE. (2020, Feb 8). Deep Learning for Image Super-resolution: p. 24.

[6] Al Basheer, O., " On Some Novel Simplex Linear Codes Defined Over The Algebraic Ring $\boldsymbol{F2+vF2}$ ", Neoma Journal Of Mathematics and Computer Science, 2023

Doi: https://doi.org/10.54216/GJMSA.080204

Received: Mach 22, 2023 Revised: May 31, 2023 Accepted: July 29, 2023