Convolutional Neural Network

Image Super Resolution (SOC2021)

Harsh Shah

July, 2021

Contents

1 Introduction

This article aims to describe the project to improve resolution of low quality images (Image Super-Resolution). The project described in this article achieves this using Convolutional Neural Networks (CNNs). Indeed there are other state-of-art algorithms (like ESRGAN, VGG loss networks, and many more) but here I would like to describe how we got hold of the problem statement and what all steps we took to tackle the problem using CNNs.

2 Platform and Softwares used

To train the CNN, we have used **Keras** (API over **tensorflow**) on Python3. The complete programming is carried out on the web-application, Jupyter Notebook along with Google Colab for having access to GPU to train the network. The code for the project as well as the saved networks are available on my public github repository.

3 Overall Plan

To carry out super resolution process, the training process involves:

- **Extracting patches and pre-processing**: As a first step, overlapping patches are extracted from low resolution image and the low resolution patches are processed using bicubic interpolation technique. The resulting output is converted to high dimensional vector.
- **Non-linear mapping**: Each high dimensional vector is mapped to another vector of same dimension using non-linear filter.
- **Reconstruction**: The patches are then combined to get the final higher resolution image.

It is clear from the SRCNN paper that all the above processes can be modeled as a convolutional layer. So let's dive into the specifics of the project.

4 Training set

Initially, we planned to gather about 100 high resolution images and then down sample them in order to get a dictionary mapping between high resolution and low resolution images. However, there already existed a tensorflow dataset named Div2K which had the exact same thing we required with scale factor 2. After loading the required dataset, we upsampled the low resolution images by the scale factor using bicubic interpolation and then extracted non-overlapping patches of dimension 50x50. This made our data set as large as about 100,000 images.

5 Network Architecture

So for building the network we have used three 2DConv layers with specifications: (9,9) channels 32, (1,1) channels 64, (5,5) channels 1 respectively. We faced problem in giving input dimension to the layer since we had decided to perform the convolutions only on luminance channel and input to a 2D conv must be 3-D. We tried different things like using a 3D conv over a 2D conv and vice-versa, using only 3D conv, and many more but ultimately ended up just resizing the input to have 3rd dimension(depth) 1.

6 Network Training

Training of the network was well documented and easily achievable in Keras/Tensorflow. We used Adam optimiser and MSE as loss function. We kept the number of epochs greater than 20 for proper training of the network. However, in order to prevent over-fitting we used a functionality of Tensorflow compile method, which stops the training when the validation error diverges and resorts back to previous epoch's weights. We trained many models experimenting different parameters: Using DCT as bases, Without DCT epochs 20, without DCT epochs 30.
However, the output of the network was 38x38 (because we did not use padding same to prevent deterioration of the image). So we down sampled the ground truth 50x50 image to 38x38 (again using bicubic interpolation). Thereafter, we trained the network on Google Colab using GPUs (cause training time is too much on a CPU). The trained network was saved and then downloaded to be used later on CPU.

7 Prediction on new images

Now that the model was ready to be used for predicting high resolution images, any new input image was first resized to the required scale and then to nearest 50 (using bicubic interpolation), then the image was broken into 50×50 patches (we have also tried using overlapping patches, checkout repository). The sub-images are passed through the network and corresponding 38×38 output images are up-sampled to 50×50 (again using bicubic interpolation). The final high resolution image is then reconstructed using the patches obtained. The complete process is neatly encapsulated in a class named "Super-Resolution".

8 Problems faced

There were many instances during the project when we faced a standstill and had to brainstorm our ideas and explore to break-through the tough spot. Some of them are listed below:

- **Getting the dataset:** After getting the gist of the project, we had to create a dataset but we were unaware that something like tensorflow datasets exist. Luckily, one of us found about Div2K dataset in a tutorial and things went ahead.
- **Reading the images:** There many subtleties involved in how different libraries treat images, and one of the major issues was the resizing of images. Numpy resize requires (height,width) as a parameter, while OpenCV resize requires (width,height) as a parameter which made debugging a strenuous task.
- **Building the network:** While building the CNN network, there were many errors for dimensions, which we could not figure out why. Later we realized that how tensorflow works, how it adds the dimension of batch size by itself what input dimensions should be given.
- **Pre-processing:** To match the dimensions of the ground truth image we interpolated the low resolution image but even after that the network reduced the dimension of the input image because we kept valid padding(same padding deteriorated the image at edges). We then used bicubic interpolation(again) to upsample the output.
- **Google Colab:** We learnt it the hard way that processes running in Google Colab GPU cannot be left unattended(else the process terminates and all progress is lost). Twice we spent hours waiting for the training process to end only to see the process terminated in between.

9 Wrapping Up

To avoid any confusion regarding the files in the github repository I would like to give some description of the files and folders in here:

- without_dct : First trained network (redundant)
- without_dct_E20_B30_I5000 : 20 Epochs, 30 batch_size, 5000 training images
- dct_E30_B32_I100000 : Network trained with DCT bases
- without_dct_E20_B30_I100000 : 20 Epochs, 30 batch_size, 100,000 training images
- Image Super Resolution.pdf : SRCNN research paper
- ImageSuperResolution(SoC).ipynb : Jupyter Notebook having the complete program(.py version is also there)
- testing_dct.ipynb : Jupyter notebook for training and test dct network is present(.py version is also there)
- sage.png : testing image(redundant)

10 Future Plans and Extrapolation

There is certainly a lot of place for improvement in the project we took up and even today better and better ways are coming up for balancing trade offs between computational time and desired result(that is, greater SSIM metric). I intend to further experiment using different convolutional layer and twitch hyperparameters to improve the results of our network, because even after performing many experiments we could not even surpass bicubic results. Of the various solutions out there, I would like to study ESRGAN and networks with SSIM loss.

11 References and links to github repositories

- Referred CNN Paper
- README.md
- First draft of helper functions
- Block patch extraction
- Tensorflow dataset
- Model Trained without DCT
- Model Network with DCT
- First Reconstruction
- CNN
- ImageSuperResolution Class
- Callbacks in Tensorflow
- Results