

# Convolutional Neural Networks Super-Resolution Project

**Mentors :**

**Sahasra Ranjan, Adarsh Raj**  
**Sibasis Nayak, Aniket Gudipaty**

**Project members (Mentees) :**

**Harsh Shah** (prodigy.harsh18@gmail.com)  
**Arpon Basu** (arpon.basu@gmail.com)  
**Shubham Kamble** (kamblesn26@gmail.com)

## Contents

<b>1</b>	<b>Some Background about CNNs</b>	<b>4</b>
<b>2</b>	<b>General Procedure of the Project</b>	<b>4</b>
<b>3</b>	<b>Detailed Code Flow</b>	<b>5</b>
<b>4</b>	<b>Individual Contributions</b>	<b>6</b>
<b>5</b>	<b>Problems faced</b>	<b>7</b>
<b>6</b>	<b>Results</b>	<b>8</b>
<b>7</b>	<b>Acknowledgements</b>	<b>8</b>
<b>8</b>	<b>Bibliography</b>	<b>13</b>

### **Abstract**

This report will summarize the our learnings from the Super-Resolution project, which includes, but is not limited to learning about image processing (using PIL, OpenCV libraries in python), Neural Networks (specifically Convolution Neural Networks), performance metrics such as PSNR (Peak Signal to Noise Ratio). We also intend this to serve as a base for anyone taking up this project of image super resolution using CNNs, to avoid the mistakes we did and to have a streamlined path for their complete ascent.

**Note:- As asked for, all our codes have been well documented in this report. In fact, we've gone a step further and hyperlinked all the pieces of code in the bibliography with descriptive names, and cited them at appropriate places wherever required.**

## 1 Some Background about CNNs

Before beginning the project, through the tutorials provided by the mentors, we had begun to develop some idea about neural networks, linear regression, accuracy metrics etc., so by the time we had to begin the actual project, we had had some initiation. To further bolster our knowledge, we were required to do a our course on Convolutional Neural Networks in Coursera, taken by professor Andrew Ng.

He explained some crucial differences between CNNs and the simple neural networks (fully connected layers of neurons that is) we had learnt so far : For simple tasks whose outputs could be neatly classified into a small number of categories, say handwriting classification (0 - 9, only 10 categories), fully connected layers of neurons were okay, but as our tasks grow bigger and bigger both in size and complexity, the number of trainable parameters in a simple model grows to very large numbers which are not easily managed. Moreover, it's also observed that deeper neural networks do not always give the best performance results, and in fact performance may even deteriorate if the networks become too deep.

Thus the need for Convolutional Neural Networks arises, which use a specific mathematical process called **convolution** to map inputs (basically **tensors**) to other tensors (the convolution process requires a filter whose values are basically our training parameters). The first 2 dimensions are usually that of the image, while the third dimension is usually something called the **number of channels**, which is quite important as will become apparent below.

The number of parameters in a CNN are usually much smaller than a corresponding simple network, and hence they're easier to train (but by no means less effective). Modern NN architectures usually combine a CNN with a simple network to get a **softmax output**.

For this project, as with most projects, as Andrew Ng himself advises, we simply implement a neural network model already articulated and implemented in a paper[1] (Ng's point of view being that one must try to accomplish one's task as best as one can by seeing if a little tweaking of hyper-parameters on an existing model suffices). The exact details of how we went about doing that is described in the sections below.

## 2 General Procedure of the Project

So according to [1] we implement a **Super-Resolution Convolutional Neural Network (SRCNN)** consisting of **3 layers**[6][7], whose jobs can be broadly described to be Patch extraction and representation[4], non-linear

mapping and reconstruction of the patches[8].

At this juncture it's also important to mention a baseline with which we will compare our performance, which is that of **bicubic interpolation**. What does this entail? Bicubic interpolation is basically a mathematical process by which a low resolution image can be **up-scaled** to a higher resolution image by interpolating (low-information) patches in the image to a high-information (resolution) one through a cubic polynomial. If one thinks about it, it's not too different from how successively better solutions of Laplace's equation are obtained in numerical analysis from an initial seed : small patches are taken, the value at the centre of the patch is replaced by the **average** of the values at the boundary, and the process is repeated till satisfaction. Now, instead of taking the average (which is a **linear** function of it's inputs), we simply replace it by a **cubic polynomial** (obtained through the Lagrange interpolation formula) and repeat the same process to get an up-scaled image.

The net point of the above digression is that since modern languages like MATLAB and Python can implement bicubic interpolation on images in  $< 1$  second, whatever neural network we choose to implement must give a performance at least as good as the bicubic one.

Thus, in our neural network, we immediately scale the input image via bicubic interpolation, and the neural network then operates on the scaled up image (to hopefully) produce a better result.

The first step of the neural network mapping is **patch extraction and representation**, which basically generates patches from our image and applies a simple ReLU filter on them. Then comes the **non-linear mapping**, which basically takes the patch and maps it to a higher dimension (this is basically where the extra information is added to the image), and finally in the last layer (**reconstruction**), all the patches from the images are collated back together to form our output image.

### 3 Detailed Code Flow

Diving straight right in, me and Harsh Shah first defined some functions we would need for our code, such as converting images to DCT and vice versa[3][4], and for patch extraction, we defined a **block patch-extraction code**[4]. The term *block* is of special significance as it divides up a picture into identical (w.r.t. dimensions), *non-overlapping* blocks of smaller images, while in *another* method of patch-extraction, one takes *overlapping* patches along with strides. The difference between the two arises after reconstruction, where the reconstructed image from block extraction looks "blocky", while the reconstructed image from the overlapping patches are relatively

smoother (we realised this after the first reconstruction though [8]). So anyways, after this we obtained our dataset [5] from Tensorflow (through `tfds.list_builders()`), which provides us 900 pairs of images (800 test and 100 validation), with each pair containing a low resolution image and it's higher resolution counterpart (scale factor = 2). All of these images were loaded onto our machines (or colaboratory GPUs), were passed through the patch extraction functions (that greatly increases the number of input images as patch sizes are quite small,  $50 \times 50$  pixels). After patch extraction we had obtained about 100,000 ( $50 \times 50$ ) images, which we used for training our network. We kept the number of epochs more than 20 so that network is trained efficiently, but at the same time we also kept an eye on validation dataset sets and their metrics to make sure that the our model is not overfitted. After passing through the network  $50 \times 50$  images what we got as output was  $38 \times 38$  images (we avoided using same padding cause it would deteriorate the output image at edges). The corresponding ground truth image ( $50 \times 50$ ) was down sampled to ( $38 \times 38$ ) to match the dimensions of output using bicubic interpolation. The loss function for training the network was MSE (**M**ean **S**quares **E**rror) and optimiser used was Adam optimiser. In order to check for overfitting of the model Keras/Tensorflow provides a parameter called "callbacks" [11] which would stop the training process if the validation error starts diverging (hyperparameters like 'patience' and 'error metrics' can be adjusted).

Now, the model [7][6] 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  $50 \times 50$  patches (we have also tried using overlapping patches, results here [12]). The sub-images are passed through the network and corresponding  $38 \times 38$  output images are up-sampled to  $50 \times 50$  (again using bicubic interpolation). The final high resolution image is then reconstructed [8] using the patches obtained. The complete process is neatly encapsulated in a class named "Super-Resolution" [10].

## 4 Individual Contributions

During the initial phase of the project where we had to go through the Coursera lecture series, all of us covered the material on our owns, though continuously animated by discussions among ourselves about the course content which deepened our understanding.

Then when the project began, we first had to define some helper functions[3][4],

which the authors Harsh Shah (from images to DCT and vice versa) and Arpon Basu (patch extraction and reconstruction functions, both dense and non-dense versions) did. Then, in the next step of the project, we had to train the neural network which we used to derive our results.

Then Arpon Basu designed the neural network architecture while Harsh Shah implemented it to produce the first prototype of the neural network [6]. The dataset [5] for training the model was found out and provided, with some help from our mentor Sahasra Ranjan, by the author Shubham Kamble.

Arpon Basu then also replicated the prototype for DCT bases here [7].

After that, we observed that using accuracy for training the networks wasn't working out properly, so Shubham Kamble suggested we use PSNR as a metric and take dense patches for patchification (we had been using non-dense patches before that).

Guided by that, Harsh Shah then trained the models [8][9][?] and produced the results attached in this report.

Arpon Basu then documented the entire code and pushed it in the repository, thus culminating in the end of the project.

## 5 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. 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 net-

work 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 Colaboratory:** 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.

## 6 Results

We present a tabulation of our results here in terms of both numerical metrics (**decibels**, which is the unit in which PSNR gives answers[12]) as well as one visual confirmation for each type of network trained for the reader to peruse through (1, 2, 3, 6). Also all networks here have been trained for a dataset of 100,000 images and a scale factor of 2, as mentioned above. This has also been included here [2].

The tables may intersperse with the flow of other sections.

Project Results		
Ground Truth	Grayscale (dB)	RGB image (dB)
Bicubic	49.63	45.42
With DCT bases (without dense patches)	17.95	— (Grayscale is already low)
Without DCT and without dense patches	28.79	28.14
Without DCT + Dense patches	29.85	28.98

## 7 Acknowledgements

We would like here to briefly acknowledge all the help we have received in our journey in this project : We would first like to thank our mentor for this project, **Sahasra Ranjan**, who was very committed and guided us ably throughout the project, even looking up for relevant pieces of code (such as



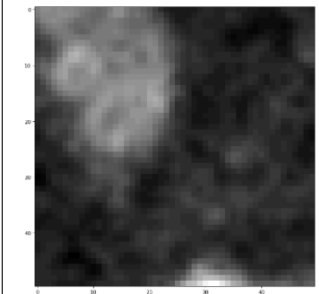
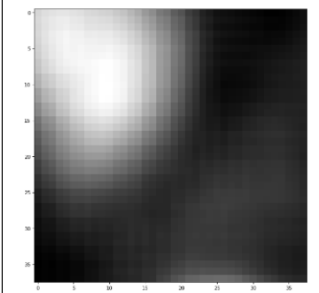
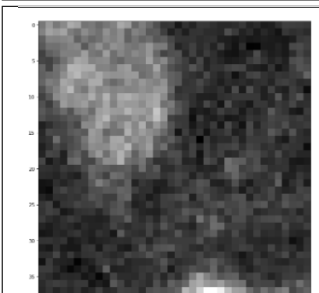
	Processed Input
	Predicted
	Ground Truth

Table 1: Without DCT + dense Patch Extraction (Before Reconstruction)

the tensorflow function `tfds.list_builders()`, and many more along the way) when we couldn't find it.

We would also like to thank the other mentors for the CNN Project, **Adarsh Raj, Aniket Gudipaty and Sibasis Nayak**, who diligently prepared quite thorough tutorials (in the learning phase) for us, before this project began, which helped us clear many of our concepts and develop a clear understanding about how things proceed in this field of neural networks.

I would also like to thank all members of WnCC who are organizing or helping in organizing this entire event, for providing such a wonderful opportunity to freshers for getting an early exposure in the various avenues that coding can lead one down to, as well as continuously supervising us along the way through READMEs and videos, thus keeping progress steady.

Finally, I would also like to apologize to anybody I may have missed on my way inadvertently.




	Bicubic Interpolation
	Predicted (Blurring happened mainly due to reconstruction)
	Ground Truth

Table 2: Without DCT + dense Patch Extraction (After Reconstruction)


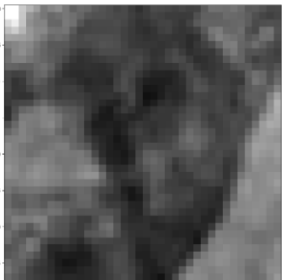
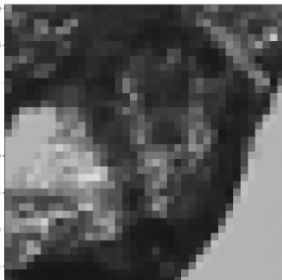
	Bicubic Interpolation
	Predicted (Notice the bright top-left corner)
	Ground truth (But down sampled from $50 \times 50$ to $38 \times 38$ to match network output dimensions)

Table 3: Using DCT bases (Only Grayscale)




	Bicubic Interpolation
	Predicted (Notice that there are distinct blocks formed due to non-dense path extraction)
	Ground Truth

Table 4: Without DCT and not using dense patch extraction

## 8 Bibliography

- [1] [Referred CNN Paper](#)
- [2] [README.md](#)
- [3] [First draft of helper functions](#)
- [4] [Block patch extraction](#)
- [5] [Tensorflow dataset](#)
- [6] [Model Trained without DCT](#)
- [7] [Model Network with DCT](#)
- [8] [First Reconstruction](#)
- [9] [CNN](#)
- [10] [ImageSuperResolution Class](#)
- [11] [Callbacks in Tensorflow](#)
- [12] [Results](#)