ASSIGNMENT-3

Mohamed Farhan

Mithun Mohan Raj

Pushkar

Group 10

Data pre-processing:

As the given test images for this assignment are of very large dimensions (more than 2000 x 2000 pixels), we have down-sampled the images to 500 x 500 pixels to speed up the training process.

(2592 x 2592 pixels)

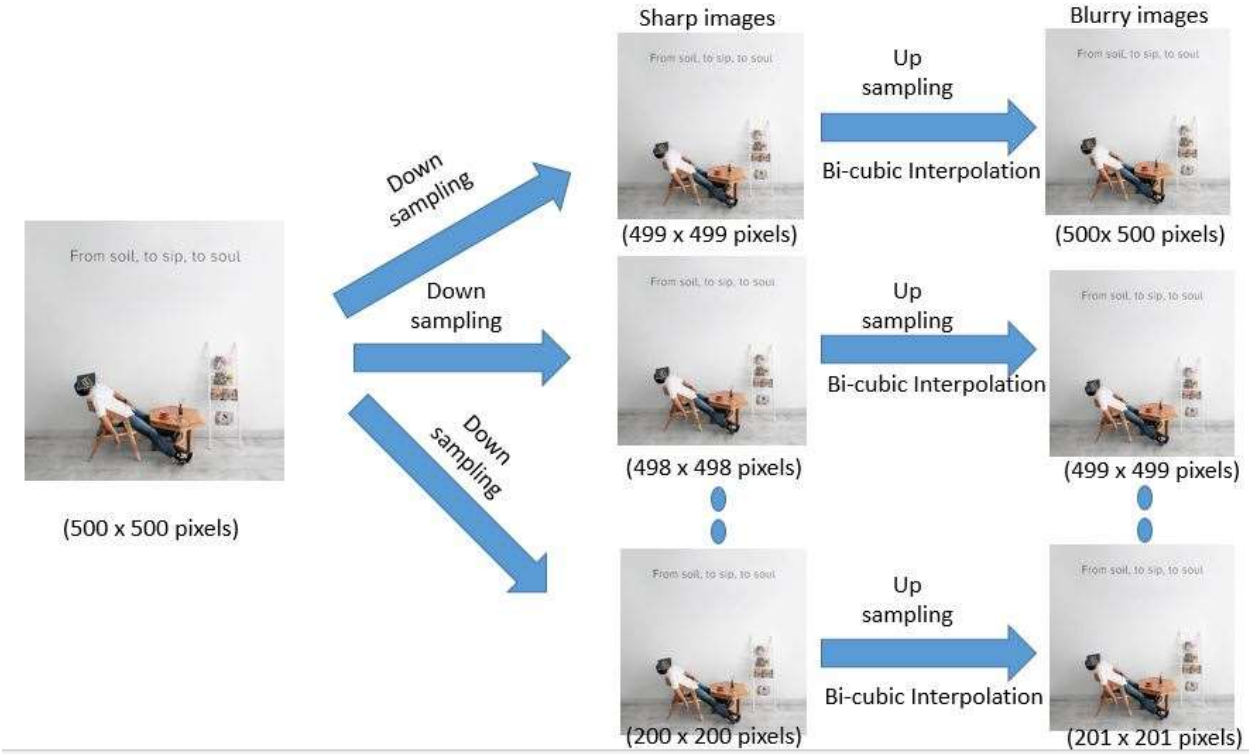
Down sampling

(500 x 500 pixels)

Creating training data:

- **From the resized 500x500 image, we again down-sample the image to get multiple sharp images of different size for the training process.**
- **We up-sample these sharp images using bi-cubic interpolation to get their equivalent blurry images that is fed into the convolutional neural network for training as input.**

The following image illustrates the above process:

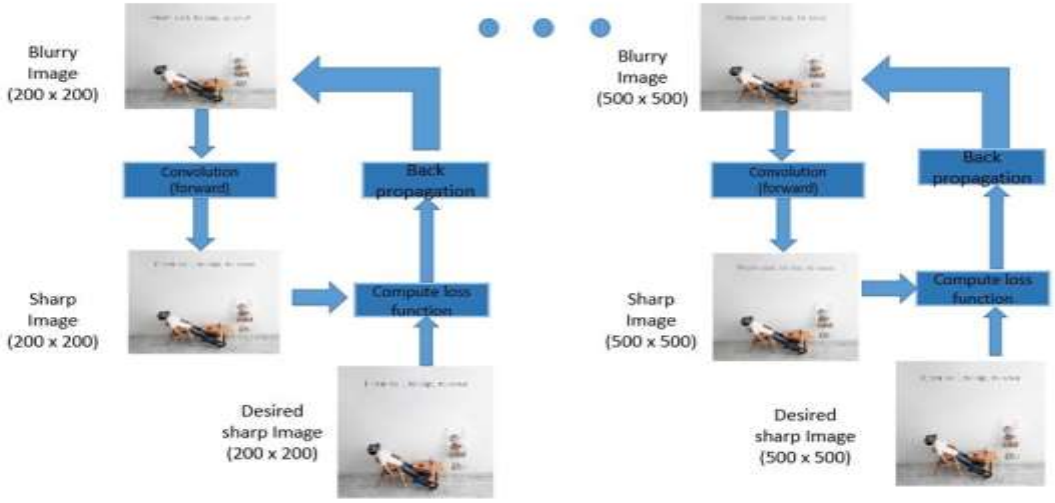


Training the Network:

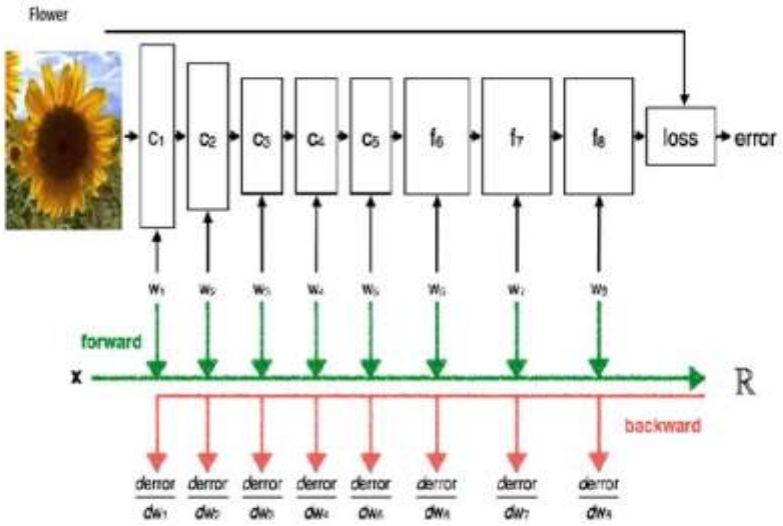
- **We feed the network with all the up-sampled blurry images one at a time, which performs the convolution operation and returns the sharp images.**
- **We calculate the loss function by finding mean square error between the sharp output image produced by the network and the desired sharp image of same dimensions.**
- **We back-propagate the loss function to the network so that network learns and updates the weights and biases for the next input image.**
- **The above process is repeated until all the images in the batch are fed into the network.**

The following images depict the whole training process:

The network is trained after each iteration through back propagation



Error gradients are computed with respect to weights in the process of back propagation



Layers used to build Convolutional Neural Networks

Convolutional layer:

- **BLURRY INPUT** [200x200x3] will hold the raw pixel values of the image, in this case an image of width 200, height 200, and with three color channels R, G, B.

- **Zero Padding:**
  Zero-padding refers to the process of symmetrically adding zeroes to the input matrix. It is required because we need to preserve the dimensions of the input image before each convolution operation.

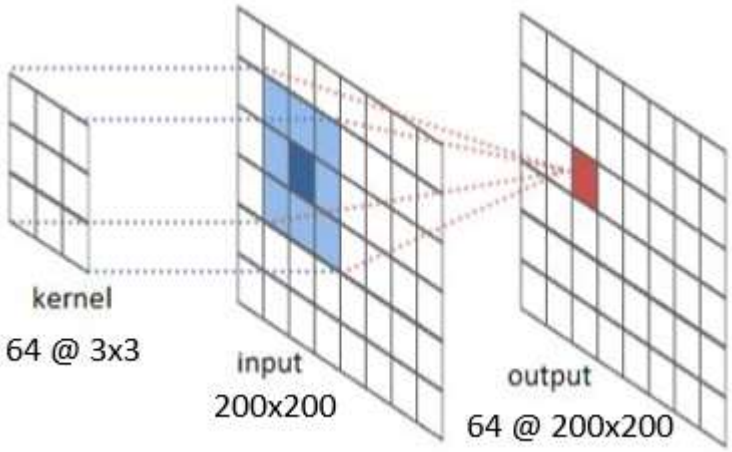
0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

- The convolutional layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.

- In our case, the filter on the first layer of the ConvNet has a size 3x3x3 (i.e. 3 pixels width and height, and 3 because images have depth 3, the color channels).

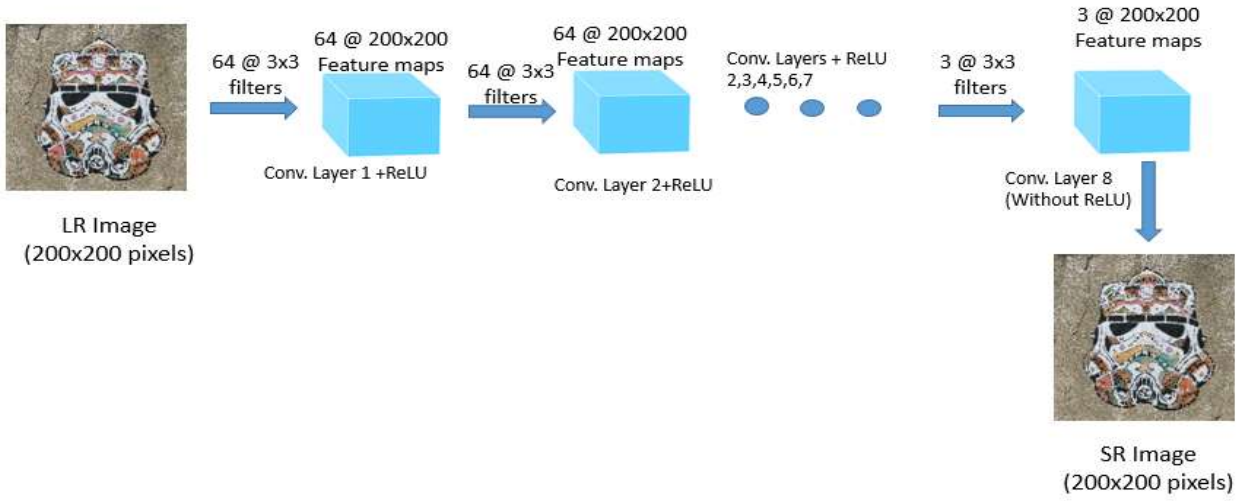
- **CONVOLUTIONAL** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

- We will have an entire set of 64 filters in each convolutional layer, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.



- We have 8 such convolutional layers where each layer has 64 filters each of dimension 3x3 except the 8th layer which has 3 filters each of dimension 3x3 to reproduce the **SHARP** color (R, G, B) image.

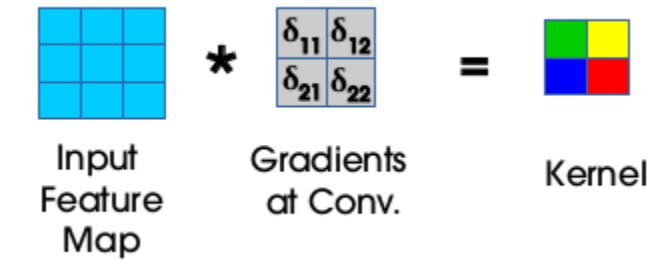
Convolutional +ReLU Layer Architecture:

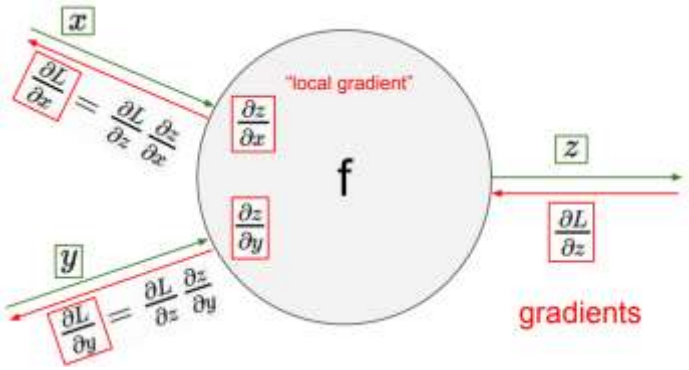


Back-propagation

We start from gradient $\partial E/\ \partial yl$ from last layer and

- **Propagate gradient back: $\partial E\ /\partial yl \rightarrow \partial E\ /\partial yl-1$**
- **Compute the gradient of E with respect to weights wl: $\partial E\ /\partial wl$**
- **For back-propagation there are two updates performed, for the weights and the gradients.**
- **We compute ∂E/∂W which can be interpreted as the measurement of how the change in a single pixel W_{mxn} in the weight kernel affects the loss function E.**





Steps to implement for backpropagating the error(only final layer)

- If we denote our predicted output estimations as vector **p**, and our actual output as vector **a**, then we can use:

$$error = J = \frac{1}{2}(\vec{p} - \vec{a})^2$$

- The derivative of the error w.r.t. the output was the first term in the error w.r.t. weight

$$\frac{d}{d\vec{p}} error = \frac{d}{d\vec{p}} J = 2 * \frac{1}{2}(\vec{p} - \vec{a})^{2-1} * 1 = (\vec{p} - \vec{a})$$

- The derivative of the error w.r.t. the weight can be written as the derivative of the error w.r.t. the output prediction multiplied by the derivative of the output prediction w.r.t. the weight

$$\frac{\partial error}{\partial W^1_{ij}} = \frac{\partial J}{\partial W^1_{ij}} = \frac{\partial J}{\partial p_j} \frac{\partial p_j}{\partial W^1_{ij}}$$

- The derivative of the output prediction w.r.t. the weight is the derivative of the output w.r.t. the input to the output layer (**p_i**) multiplied by the derivative of that value w.r.t. the weight

$$\frac{\partial p_j}{\partial W^1_{ij}} = \frac{\partial p_j}{\partial p_i} \frac{\partial p_i}{\partial W^1_{ij}}$$

- **p_i** is a summation of each weight multiplied by each **z_j**
- Take the derivative of **p_i** with any arbitrary **z_j**, the result would be the connecting weight

$$\frac{\partial p_i}{\partial z_j} = W^3{}_{ij}$$

$$\frac{\partial p_i}{\partial W^1{}_{ij}} = W^3{}_{ij} * \frac{\partial z_j}{\partial W^1{}_{ij}}$$

The above steps are repeated for all the layers to calculate the all gradients

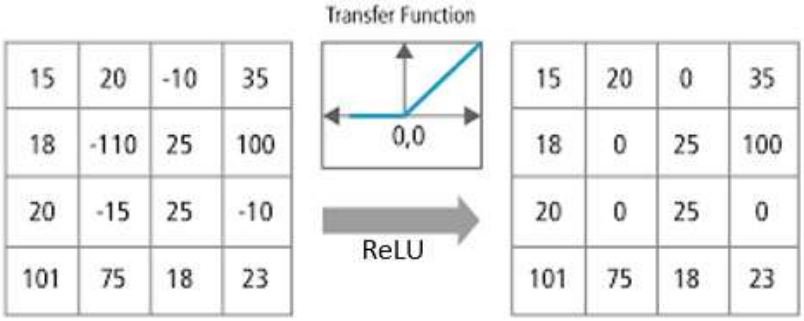
After calculating the gradients for all layers, we perform the weight and bias updates using the following formula:

$$W := W - \alpha \frac{\partial J}{\partial W}$$

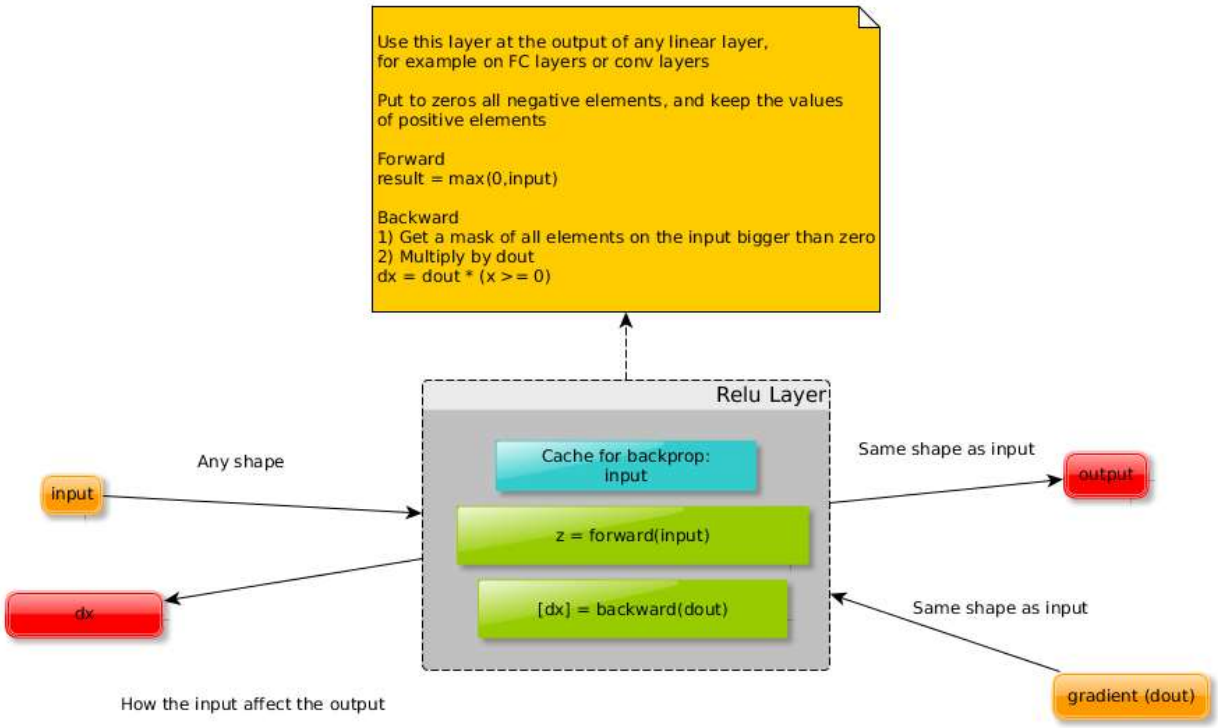
This is the gradient descent **update rule.** It tells us how to update the weights of our network to get us closer to the minimum we're looking for.

Rectified Linear Unit (ReLU):

- **ReLU** layer will apply an element wise activation function, such as the max(0,x), thresholding at zero. This leaves the size of the volume unchanged.



- The output of each convolutional layer is passed through a ReLU layer.
- The output of the 8th layer is **NOT** passed through any ReLU layer.



Loss Function

Error:

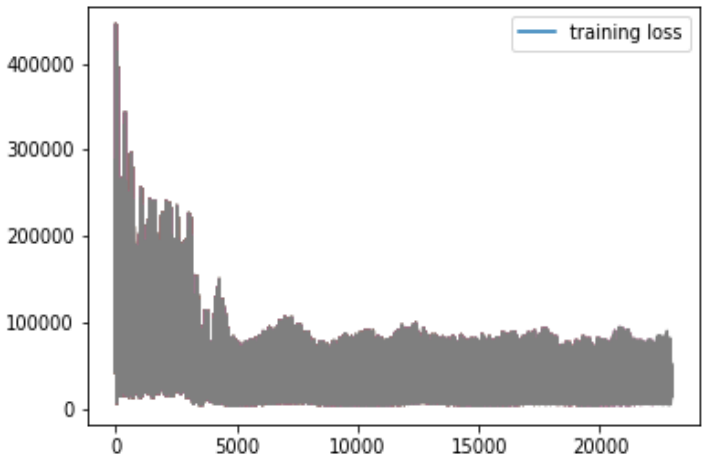
For a total of P predictions, the predicted network outputs y_p and their corresponding targeted values t_p, the mean squared error is given by:

$$E = \frac{1}{2} \sum_{p} (t_p - y_p)^2$$

Learning will be achieved by adjusting the weights such that y_p is as close as possible or equals to corresponding tp.

In the classical back-propagation algorithm, the weights are changed according to the gradient descent direction of an error surface E.

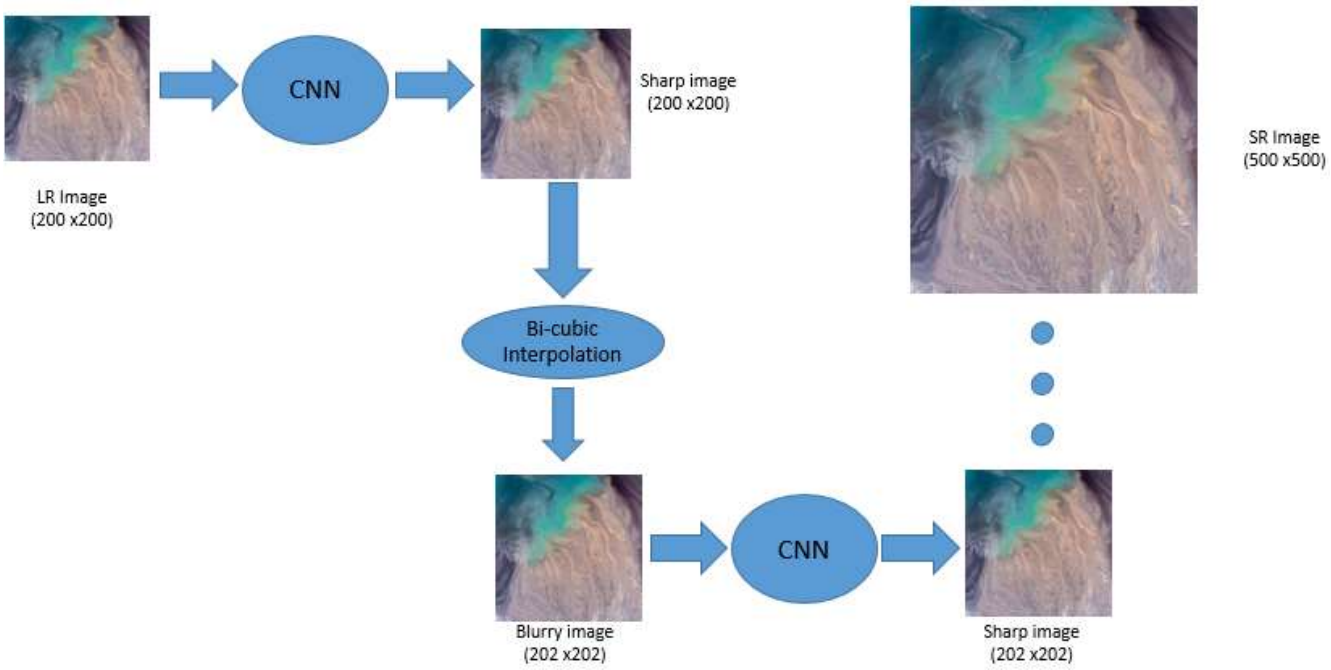
The following graph is obtained by plotting the graph on training loss Vs number of epochs (iterations):



- The output of the 8th layer is the predicted output sharp image of dimensions n x n which is compared with the desired sharp image of same dimensions.
- The calculated loss function is used for calculating the gradient of cost with respect to the output to back-propagate the error.

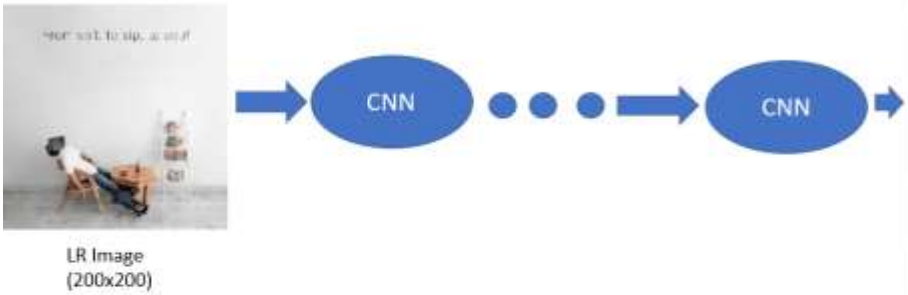
Testing the network:

- **Low resolution (LR) images are passed through the convolutional neural network to obtain its corresponding sharp images.**
- **These sharp images are up-sampled using bi-cubic interpolation to obtain its corresponding up-sampled blurry images.**
- **These up-sampled blurry images are once again passed through the convolutional neural network to obtain its equivalent sharp images.**
- **This process is repeated for as many iterations as required to obtain the desired super resolution image.**
- **This technique is well illustrated below with the help of a block diagram.**



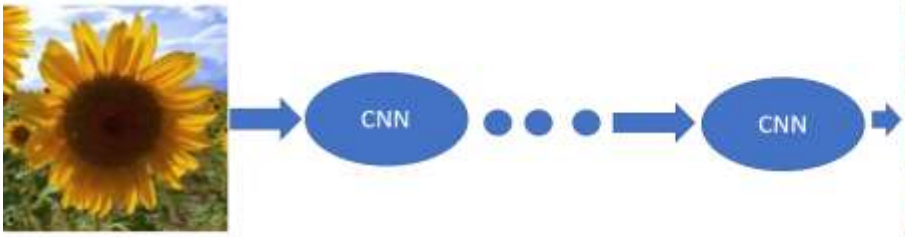
Results:

LR-Image to SR Image:



LR Image
(200x200)

SR Image
(500x500)



LR Image
(200x200)

SR Image
(500x500)

LR Image
(200x200)

CNN ● ● ● CNN

SR Image
(500x500)

LR Image
(200x200)

CNN ● ● ● CNN

SR Image
(500x500)