Buildings built in minutes - SfM and NeRf

Shreyas Kanjalkar
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, USA
skanjalkar@wpi.edu

Khizar Mohammed Amjed Mohamed
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, USA
kamjedmohamed@wpi.edu

***Abstract*—In this project we implement an entire 3D rigid structure from a set of images with different viewpoints. We implement a classical computer visin approach and a deep learning approach using the Original NeRF method**

***Index Terms*—component, formatting, style, styling, insert**

I. Phase-1

A. Introduction

The pipeline for this project is as follows:

1) Feature matching and outlier rejection using RANSAC and fundamental matrix
2) Estimate the Fundamental matrix from initial two views
3) Estimate the Essential matrix from Fundamental matrix
4) Estimate the camera pose from the Essential matrix
5) Camera pose disambiguation to find the correct camera pose of the second view using Cheirality and Linear Triangulation
6) Refining views formed from Linear Triangulation using Non-Linear Triangulation
7) Perspective n-points to register the new image, new points, new camera pose relatively into the model
8) Bundle adjustment to optimize the entire view from all the points from the cameras

B. Fundamental matrix

Let us consider Figure 1 where X is a common point in 3D space between two camera poses. The point X is projected onto C as x and onto the image plane C' as x'. The fundamental matrix F draws a relationship between x and x'. In this report we shall avoid going too much into detail with regards to derivations. From **?** we arrive at the following equation.

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{32} & f_{32} & f_{33} \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

x,y and x',y' correspond to the pixel coordinates of the points x and x'. F can be solved using an eight point algorithm. Therefore given m correspondences (m is atleast 8) points we can write the above equation as follows.

Now that we have established the procedure to compute the fundamental matrix we need to make sure that our point correspondences are accurate while calculating the fundamental matrix. We need to make sure that we reject outliers. This

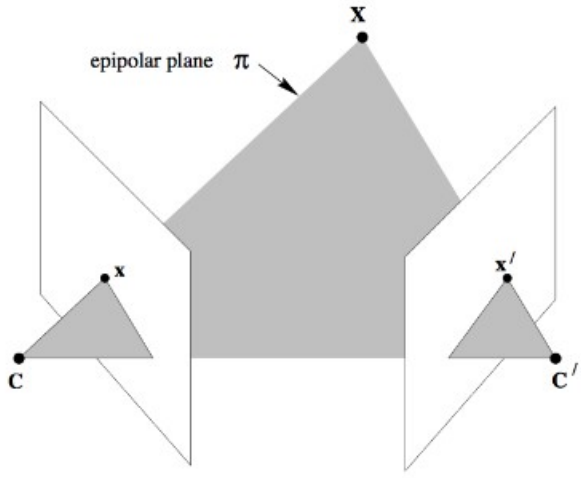

Fig. 1: Representation of a common point in two image frames

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx_m' & x_my_m' & x_m & y_mx_m' & y_my_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

Fig. 2: Calculating fundamental matrix using SVD

can be using RANSAC, where we find the best F matrix. The best F matrix, when substituted in equation 1 should return a value close to zero for the equation. The following RANSAC algorithm used is mentioned below.

C. Essential Matrix

The fundamental matrix includes pixel coordinates,whereas in the Essential matrix, the represented points are in "normalized image coordinates".The normalized image coordinates have the origin at the optical center of the image, and the x and y coordinates are normalized by F_x and F_y respectively, so that they are dimensionless. The essential matrix can be written as follows.

n=0;
for $i = 1{:}M$ **do**
  // Choose 8 correspondences, $\hat{x}_1$ and $\hat{x}_2$ randomly
  F = EstimateFundmentalMatrix($\hat{x}_1$, $\hat{x}_2$);
  $\mathcal{S} = \emptyset$;
  for $j = 1{:}N$ **do**
    if $\mid x_{2j}^T F x_{1j} \mid < \epsilon$ **then**
      $\mathcal{S} = \mathcal{S} \cup \{j\}$
    end
  end
  if $n < \mid \mathcal{S} \mid$ **then**
    $n = \mid \mathcal{S} \mid$;
    $\mathcal{S}_{in} = \mathcal{S}$
  end
end

Fig. 3: Calculating Fundamental matrix using RANSAC

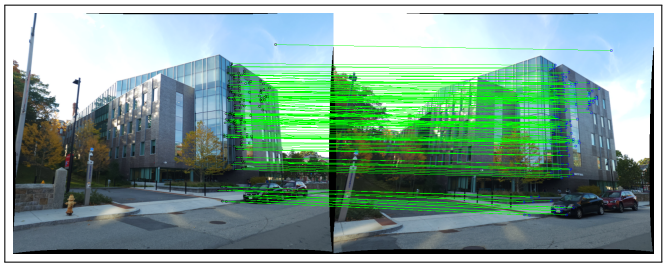

Fig. 4: Inliers at 0.01 threshold and 10000 iterations

$$E = K^T F K \tag{2}$$

Here K is the camera calibration matrix and F is the Fundamental matrix calculated from the previous section. It is important to correct for the noise from K while calculating E. This can be done by calculating the SVD of E and then reconstructing it with singular values (1,1,0) as shown in the below equation.

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{3}$$

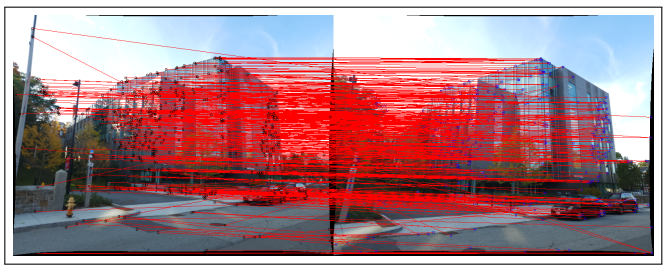

Fig. 5: Outliers at 0.01 threshold, 10000 iterations

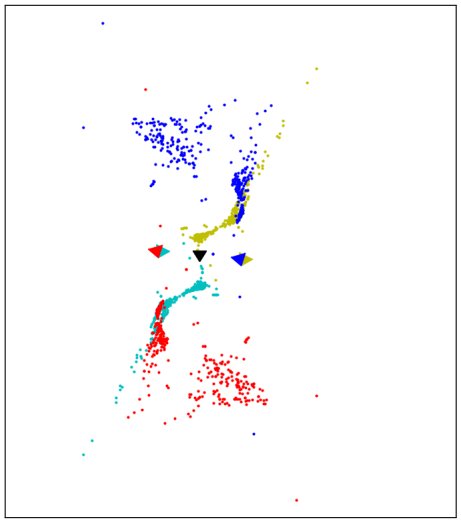

Fig. 6: Linear Triangulation of 4 poses

D. Camera pose calculation

A camera pose can be defined by using the translation vector T $\in R^{3\times1}$ and the rotation matrix R$\in$ SO(3). These can be obtained from the Essential matrix. In this report we will not derive the required equations.

$$E = UDV^T = T_x R \tag{4}$$

$$T = \pm u_3 \tag{5}$$

$$R = U \begin{bmatrix} 0 & \pm1 & 0 \\ \pm1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T \tag{6}$$

Here U and V are obtained from the SVD of E. u_3 refers to third column of U. Given that we have two solutions for R and T, we will have four camera poses in total.

E. Linear Triangulation

From the previous section we were able to calculate the rotation and translation matrices for both cameras. Knowing the projection matrix we should be able to calculate the coordinates of a point. Consider Figure 1, we know that

$$x = PX \tag{7}$$

$$x' = P'X \tag{8}$$

where P and P' are the projection matrices. In a perfect world triangulating using the above two equations should have a similar X value. However due to noise we have to use linear least squares in SVD to get the X value. The process is as follows. Since x and PX are the same. The following is true

$$x_\times PX = 0 \tag{9}$$

Fig. 7: Linear Triangulation

$$x'_{\times} P' X = 0 \tag{10}$$

These equations can be re-written as follows and solved with SVD.

$$AX = 0 \tag{11}$$

$$A = \begin{bmatrix} x_{\times} P \\ x'_{\times} P' \end{bmatrix} \tag{12}$$

Therefore using Linear triangulation we get an initial estimate of X.

F. Disambiguating camera poses

From section C and section D we have estimated X, and two unique Rotation matrices and translation matrices for each camera. There can only be one unique camera pose, therefore we can only have one Rotation vector and one translation vector for each camera. This is can be accomplished by checking for the cheirality condition for all combinations of R and T. The cheirality condition states that the reconstructed points X, must be in front of the cameras.A 3D point X is in front of the camera if the following condition is satisfied.

$$r_3(X - C) > 0 \tag{13}$$

where:
r_3 and is the third row of the rotation matrix R
C and the camera pose (the translation vector T)

Not all our triangulated points will satisfy the cheirality condition. Therefore, the best camera configuration, (C,R,X) is the one that produces the maximum number of points satisfying the cheirality condition.

G. Non Linear Triangulation

Given the camera poses and estimated 3D points(X) using linear triangulation we can further refine X. This is done by reprojecting X onto the image plane and calaculating the geometric error as follows.

$$\epsilon_{geometric} = \|\hat{x} - x\|^2 + \|\hat{x}' - x'\|^2 \tag{14}$$

Here $\hat{x}$,$\hat{x}'$ and are the re-projected image points for camera 1 and camera 2 respectively and x, x' are the actual image pairs.We can easily calculate the re-projected image points using the below equation.We use scipy.optimize.leastsq to refine our X in this section.

$$\hat{x} = P_1 X \tag{15}$$

$$\hat{x}' = P_2 X \tag{16}$$

H. Linear Perspective N point algorithm(PNP)

The goal of PnP is to estimate the pose of a camera (Rotation R, and Translation T) given a set of 3D points and their 2D correspondences in a picture captured by the camera. The following process is explained below.

Consider equation 9. It can be expanded as follows.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{\times} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} \tilde{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

In the above equation P_1, P_2, P_3 are the rows of the projection matrix of the given camera P. $\tilde{X}$ is $[X, Y, Z, 1]^T$. The above equation can be rewritten in a way so that the projection matrix terms are all towards the right-hand side.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{\times} \begin{bmatrix} \tilde{X}^T & O_{1\times4} & O_{1\times4} \\ O_{1\times4} & \tilde{X}^T & O_{1\times4} \\ O_{1\times4} & O_{1\times4} & \tilde{X}^T \end{bmatrix} \begin{bmatrix} P_1^T \\ P_1^T \\ P_1^T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

The above equation can be easily solved using SVD. It is important to note that the the projection matrix has 12 unknowns. Each 2D 3D correspondence has two constraints. Therefore using 6 points we can get 12 equations and 12 unknowns. Below how is how our equation will need to look like to solve for SVD.

$$A_{Size:18\times12} \begin{bmatrix} P_1^T \\ P_1^T \\ P_1^T \end{bmatrix}_{Size:12\times1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_{Size:18\times1} \tag{19}$$

We know the equation of the projection matrix, and it can be rewritten as follows

$$R = K^{-1} P_{1:3} \tag{20}$$

where $P_{1:3}$ represents the first three columns of the projection matrix. However, we know that R is orthonormal, and to reinforce this condition we use SVD on R again.

$n = 0$
for $i = 1{:}M$ **do**
  // Choose 6 correspondences, $\hat{X}$ and $\hat{x}$, randomly
  [C R] = LinearPnP($\hat{X}$, $\hat{x}$, K);
  $\mathcal{S} = \emptyset$;
  for $j = 1{:}N$ **do**
    // Measure Reprojection error
    $e = \left(u - \frac{P_1^T \tilde{X}}{P_3^T \tilde{X}}\right)^2 + \left(v - \frac{P_2^T \tilde{X}}{P_3^T \tilde{X}}\right)^2$;
    if $e < \epsilon_r$ **then**
      $\mathcal{S} = \mathcal{S} \cup \{j\}$
    end
  end
  if $n < |\mathcal{S}|$ **then**
    $n =| \mathcal{S} |$;
    $\mathcal{S}_{in} = \mathcal{S}$
  end
end

Fig. 8: Using RANSAC for PnP

$$SVD(R) = UDV^T \tag{21}$$

Then R can be rewritten as

$$R = UV^T \tag{22}$$

$$T = K^{-1} P_{4^{th}column}/d_1 \tag{23}$$

In the above d_1 is the leading eigenvalue of D. There are going to be errors in Pnp due to the presence of outliers. We use the RANSAC algorithm in Figure 4 to make our camera pose estimation more robust.

I. Non Linear PnP

This section is very similar to what we did in non-linear triangulation. We can refine the estimated camera pose using the 3D points and their 2D correspondences. This is done by re-projecting X onto the image plane and calculating the geometric error as follows.

$$\epsilon_{geometric} = \|\hat{x} - x\|^2 \tag{24}$$

Here $\hat{x}$ is the re-projected image points and x is the actual image point. We can easily calculate the re-projected image points using the below equation. We use scipy.optimize.leastsq to refine our camera pose parameters in this section. It is important to note that we use quaternions to represent our rotation matrices while optimizing, this is to maintain orthogonality. However, we reconvert it back to the rotation matrix when we calculate the projection matrix.

J. Non-Linear PNP and Bundle adjustment

These sections are very similar to the ones in Non linear triangulation. In the former we optimize for the cameara pose and rotation matrix , and in the latter we optimize for all 3D points and camera poses

K. Results

In figure 1 we have shown s sample of visual SFM. Our camearas are close to where our red dot is presnet in the picture below. In the following table we have shown the reproejction errors for all different steps.

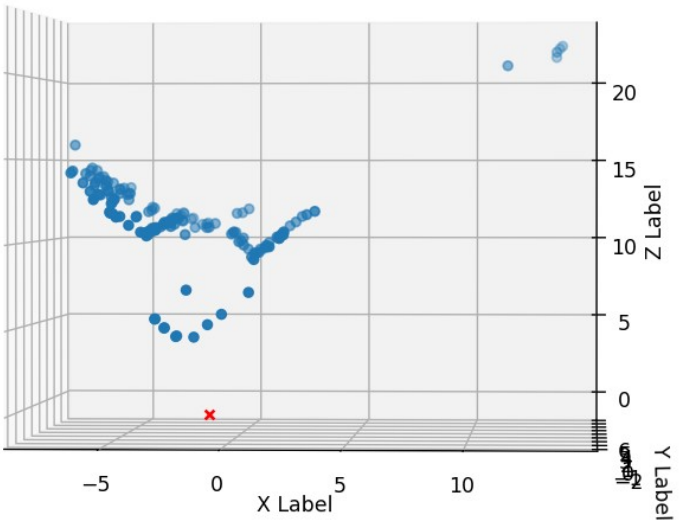

Fig. 9: 3D points represented after non-linear trinagulation

Camera set	Linear Triangulation	Non-linear Triangulation	Linear PNP	Non-Linear PNP	Bundle adjustment
`1			-	-	-
2	1.709	1.706	-	-	-
3	183.00	183.00	200.014	149.149	2.23
4	64.22	64.22	285.103	166.531	8.76
5	89.70	89.70	211.51	60.813	1.49

Fig. 10: Mean reprojection errors

II. PHASE-2 NERF

A. Introduction

This section includes details about the state of the art NeRF paper[1]. NeRF addressed the long standing issue of generating synthetic novel views from given set of images by optimizing an underlying cuntinuious volumentric scene function. The algorithm presented in the paper usesa full connected deep network, whose input is a single continuous 5D coordinate function and the output of it is the volume density and view-dependent emitted radiance at that spatial location.

The 5D function makes use of θ, ϕ which are the viewing directions at each point x, y, z in space. The pipeline to render the viewpoints are:

1) March camera rays through the scene to generate a sampled set of 3D points.
2) Use those points and their corresponding 2D viewing directions as input to the neural network to produce an amount set of colors and densitites
3) Use classical volume rendering techniques to accumulate those colors and densities into a 2D image

B. Approach

The approach inherits the benefits of voumentric represnetations: both that can represent complex real-world geometry and appearance and are well suited for gradient-based optimization using projected images. Their modethod overcomes the prohibitve storage costs of discretized voxel grids when modelling complex scenes at high-resolutions. In short their approach is given as:

1) Approach for representing continuous scnes with complex geometry and materials as 5D neural radiance fields, parameterized as basic MLP networks
2) A differentiable rendering procedure based on classical volume rendering techniques, which we use to optimize these representations from standard RGB images. This includes a heirarchical sampling strategy to allocate the MLP's capacity towards space with visible scene content
3) Positional encoding to map each input 5D coordinate into a higher dimensional space, which enables to successfully optimize neural radiance fields to represent high frequency scene content as MLP are biased towards lower frequency things

C. Neural Radiance Field Scene Representation

The continuous scene is represented as 5D vector valued function whose input is a 3D location given as $\mathbf{x} = (x, y, z)$ and 2D viewing direction (θ, ϕ) and whose output is an emitted color $c = (r, g, b)$ and volume density σ. The 5D function is a ray which passes through the pixel of the image and can be represented as:

$$\vec{r} = \vec{o} + (\vec{d})t$$

where r is the ray, o is the origin vector of the ray, d is the direction vector and t is the depth value.

D. NeRF MLP

Each sample point is of 5 dimensions. The spatial location of the point is a 3D vector of (x, y, z) and the direction of the point is a 2D vector (θ, ϕ). The authors advocate expressing the viewing direction as a 3D Cartesian unit vector d. These 5D points serve as the input to the MLP. This field of ray with 5D points is referred to as the neural radiance field in the paper.

The MLP network predicts each input point's color c and volume density σ. Color refers to the (r, g, b) content of the point. The volume density can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at that point.

The architecture is shown in the figure 5. The overall architecture is quite simple, which uses fully connected layers.

E. Implementation

From the given dataset, we read the images for train, test and validation. Using the camera_angle_x, image width and height we find the focal length of the camera. It is assumed the focal length in x and y is the same. The dataset also contains the transformation matrix for each image. Using the transformation matrix, the camera to world pose is obtained which are used for the ray directions. The origin for each ray in the image is assumed to be one point (pin hole model). The origin is shifted to the 3 entries of the last column to get the x, y and z for the origin.

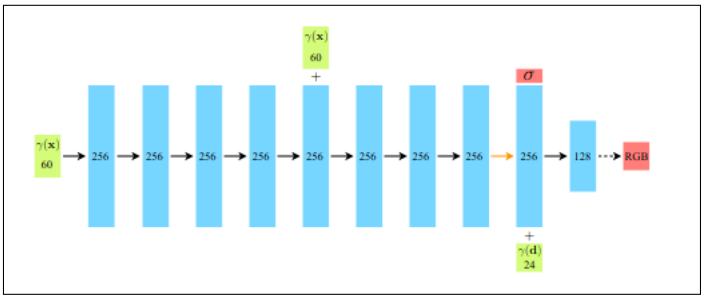

Fig. 11: Architecture of NeRF

The implementation followed the paper's directions. Instead of going through all the rays of each image(which is basically $(800 \times 800 \times 32)$. That is a lot of computation and the program runs out of gpu memory. Instead for each image we randomly sample 1024 rays. That reduces the computation as well as for the first 500 images, the image is cropped from $(800 \times 800)$to(100×100) around the center. This trains the model well for the first 500 iterations, and then includes rays from the entire image.

In order to improve the rendering, the authors proposed two methods. Positional encoding and Hierarchical sampling. The implementation also includes a coarse model and fine model. The coarse model samples points uniformly. Using the probability distribution function, each point has a weight assigned to it. Using these points, the points for new model are sampled around the points which have higher weights. This ensures that the model learns better rgb value of the image, which helps in reducing loss and increasing the PSNR. In the figures 5, 6; N_c refer to the coarse points, and N_f refer to the fine points.

F. Results

Training was done on the given dataset with perfect poses for each image. We ran the model for 200000 iterations and found the results matching with the ones matching in the paper. Fig 7 shows the plot of iterations with plots and iterations. The testing is done on the given data set at every 10000 iterations. The video of the same can be found here.

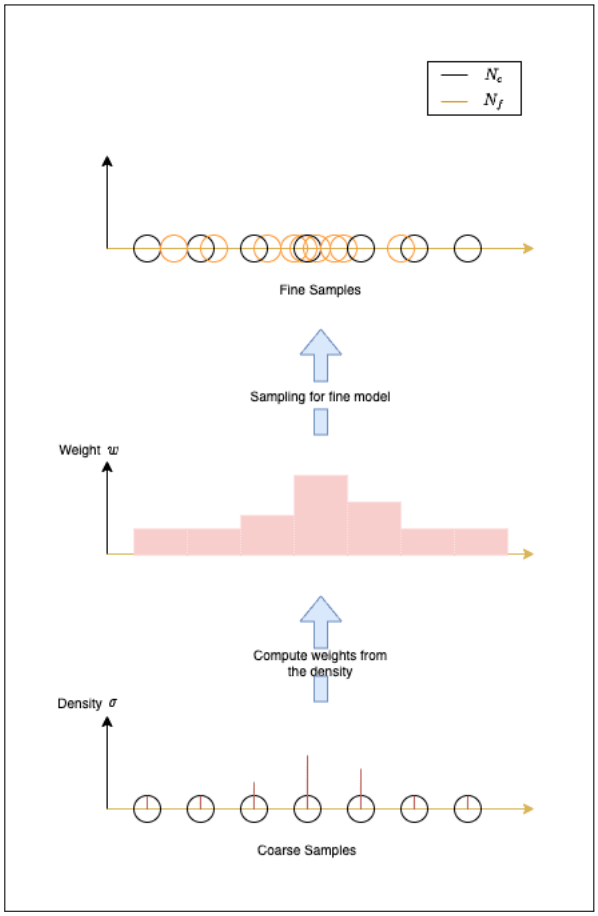

Fig. 12: Coarse Points

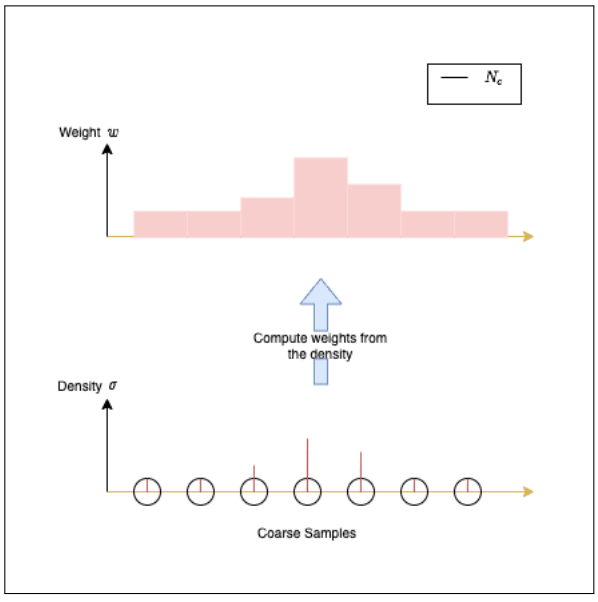

Fig. 13: Fine points sampled using coarse points

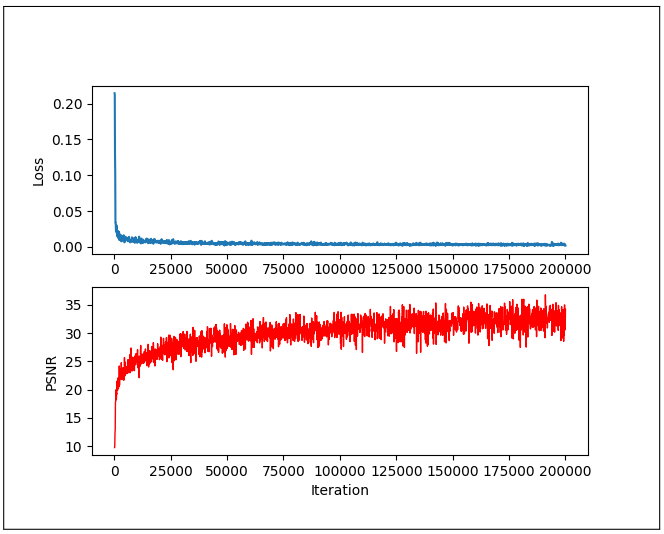

Fig. 14: Loss and PSNR for 200000 Iterations

Fig. 15: Lego fig at 1000, 10000 iteration

Fig. 16: Lego fig at 25000, 50000 iteration

Fig. 17: Lego fig at 100000, 200000 iteration