1. 加载数据

load\_llff\_data函数，默认加载llff数据。llff数据包含了poses\_bounds.npy文件，储存了姿态和最近最远处深度信息，是一个20x17（20张图片）的数组。

首先通过\_load\_data函数将其变形后得到每张图片对应的pose matrix（3x5x20）（The pose matrix is a 3x4 camera-to-world affine transform concatenated with a 3x1 column [image height, image width, focal length]）和bds（2x20）。然后用factor参数对图片进行了缩小操作。读取了images\_8内的图片（378x504x3x20）。\_load\_data函数返回poses matrix，bds，imgs三个张量。

因为从相机坐标系出发的三个轴为[-y,x,z]，需要对旋转矩阵修正，通过poses = np.concatenate([poses[:, 1:2, :], -poses[:, 0:1, :], poses[:, 2:, :]], 1)语句调整了一下顺序。然后修改了各个张量轴的顺序（move variable dim to axis 0）。然后通过bd\_factor做了尺度变换。

然后进行recenter操作，返回一个20x3x5的poses张量。根据poses得到c2w矩阵（3x5）。根据bds的最小值和最大值计算出focus depth。

得到圆形路径的各个参数，通过render\_path\_spiral函数得到圆形路径上的120个pose matrix。dists、i\_test两个参数。

然后转换了一下hwf的数据类型。

二、创建nerf模型。

1、position encoding

nerf函数的输入为位置和角度信息，实验中发现直接将位置和角度作为输入得到的结果比较模糊（fig4）。position encoding将位置信息映射到高频，提升了清晰度效果。

主要参数有max\_freq（即论文中的L）和N\_freqs。编码函数为lambda x, p\_fn=p\_fn, freq=freq : p\_fn(x \* freq)，返回编码后的embed\_fns和输出的维数，每次编码输出都增加三个维数。对于坐标L取10，对于方向L取4.

2、nerf模型。

将连续场景表示为5D向量值函数，其输入为3D位置x =（x，y，z）和2D观看方向（θ，φ），其输出是发射的颜色c =（r，g，b）和体积密度σ。其中，体积密度σ仅作为位置x的函数来预测，RGB颜色c作为位置和观察方向的函数来预测。MLP FΘ首先处理输入的具有8个完全连接层的3D坐标x（使用ReLU激活和每层256个通道），并输出σ和256维特征向量。然后，该特征向量与相机光线的观察方向相连，并传递到4个额外的完全连接层（使用ReLU激活和每层128个通道），实际代码里只实现了一层？这些层输出与视图相关的RGB颜色。最终输出RGB颜色和alpha。

作者采用了一种“coarse to fine" 的形式，创建了coarse网络和fine网络。

即render\_kwargs\_train, render\_kwargs\_test, start, grad\_vars, model = create\_nerf(args)，包含了训练集，测试集及可训练变量的信息。

然后创建了优化器，设置了学习率及学习率衰减的参数。

三、渲染及训练，render rays

get\_rays\_np函数，通过图片的宽，高，相机焦距和pose matrix得到图片上每个像素点对应的射线的起点向量和方向向量。变形得到尺寸为[(N-1)\*H\*W, ro+rd+rgb, 3]的一个张量。包含了N张图片上像素点的RGB信息和对应射线的起点和方向向量信息。

在使用ray batch的情况下，采样N\_rand个射线，得到batch\_rays, target\_s，分别包含ray batch中射线的起点，方向，编号信息。和对应编号像素点的RGB信息，用于计算损失函数。

然后调用render函数，根据ray batch预测颜色，体积密度。Render函数中，首先将射线的起点，方向，近边界，远边界合成成为了一个张量。然后进入batchify\_rays函数，在minibatch进行Volumetric rendering。

render\_rays函数，首先获取了rays\_o, rays\_d，near，far这几个参数，然后设置了采样位置，（t\_vals = tf.linspace(0., 1., N\_samples)， z\_vals = near \* (1.-t\_vals) + far \* (t\_vals)）。根据采样位置得到对应空间点坐标pts = rays\_o[..., None, :] + rays\_d[..., None, :] \* z\_vals[..., :, None]，即r(t)=o+td.

通过raw = network\_query\_fn(pts, viewdirs, network\_fn)得到网络输出。尺寸为[N\_rays, N\_samples, 4]。包含RGB信息和体积密度，即论文中的和。然后通过raw2outputs函数将初始输出转换为输出。

首先计算dists即邻近两个采样点之间的距离。乘以ray\_d向量的范数转化为真实世界距离。然后通过rgb = tf.math.sigmoid(raw[..., :3])从原始数据获取rgb信息，语句，alpha = raw2alpha(raw[..., 3] + noise, dists)，函数raw2alpha，通过原始数据中的density计算得到alpha，根据论文，即,其中为相邻两采样点间距离。要计算,还需计算系数，根据论文，,根据语句tf.math.cumprod(1.-alpha + 1e-10, axis=-1, exclusive=True)得到，cumprod()用于计算输入张量的累加积。根据论文的计算公式，即语句rgb\_map = tf.reduce\_sum(weights[..., None] \* rgb, axis=-2)得到了rgb\_map。以及depth\_map，acc\_map等输出。

得到输出之后进入优化过程，计算rgb\_map和target\_s的L2损失，训练网络。