Exercises Week 4

Task 1

```
# TASK 1: Sample n_samples linearly between `near` and `far`
# HINT: USE torch.linspace, get device with rays_o.device
distances = torch.linspace(near, far, n_samples, device=rays_o.device)

if perturb:
    mids = calculate_mids(distances)
    far = torch.concat([mids, distances[-1].unsqueeze(0)], dim=-1)
    near = torch.concat([distances[1].unsqueeze(0), mids], dim=-1)

    t_rand = torch.rand([n_samples], device=distances.device) # random numbers between 0 and 1
    distances = near + (far - near) * t_rand

distances = distances.expand(list(rays_o.shape[:-1]) + [n_samples])

# Apply scale from `rays_d` and offset from `rays_o` to samples
# pts: (width, height, n_samples, 3)
pts = rays_o[..., None, :] + rays_d[..., None, :] * distances[..., :, None]
return pts, distances
```

Task 2

```
# TASK 2: Complete the implementation of the Embedder
for freq in freq_bands:
    self.embed_fns.append(lambda x, freq=freq: torch.sin(freq * x)) # HINT: use torch.sin
    self.embed_fns.append(lambda x, freq=freq: torch.cos(freq * x)) # HINT: use torch.cos
```

Task 3

The forward function was completed, see code on the next page, and a video was rendered using the pre-trained model. In task 4 we will compare this render with the one using constant viewing direction, as their difference is worth discussing.

But as you can see to the right, the render without constant viewing direction is of decent quality and detail. This 300x300 render took around 2 minutes on one of the HPC's gpu's.

```python
def forward(self, x, viewdirs=None):
    # TASK 3: Implement the NeRF forward.
    # Make sure you understand the __init__ function first
    # NOTE: do not change the names in the init function

    input_pts = x
    input_views = viewdirs

    h = input_pts
    # for each layer with index i
    for i, l in enumerate(self.pts_linears):
        h = l(h) # HINT: feed h to the layer i and rewrite to h
        h = F.relu(h) # HINT: use relu
        if i in self.skips:
            h = torch.cat([input_pts, h], dim=-1) # implement skip with torch.cat

    if self.d_viewdirs is not None:
        alpha = self.alpha_linear(h) # HINT: feed h to alpha linear
        feature = self.feature_linear(h) # HINT: feed h to feature linear
        h = torch.cat([feature, input_views], dim=-1) # HINT: concat feature and input_view

        for i, l in enumerate(self.views_linears):
            h = l(h) # HINT: forward for views_linears of i
            h = F.relu(h) # HINT: Use relu

        rgb = self.rgb_linear(h) # HINT: calculate rgb values with rgb_layer
        outputs = torch.cat([rgb, alpha], dim=-1) # HINT: concat rgb and alpha
    else:
        outputs = self.output_linear(h)

    return outputs
```

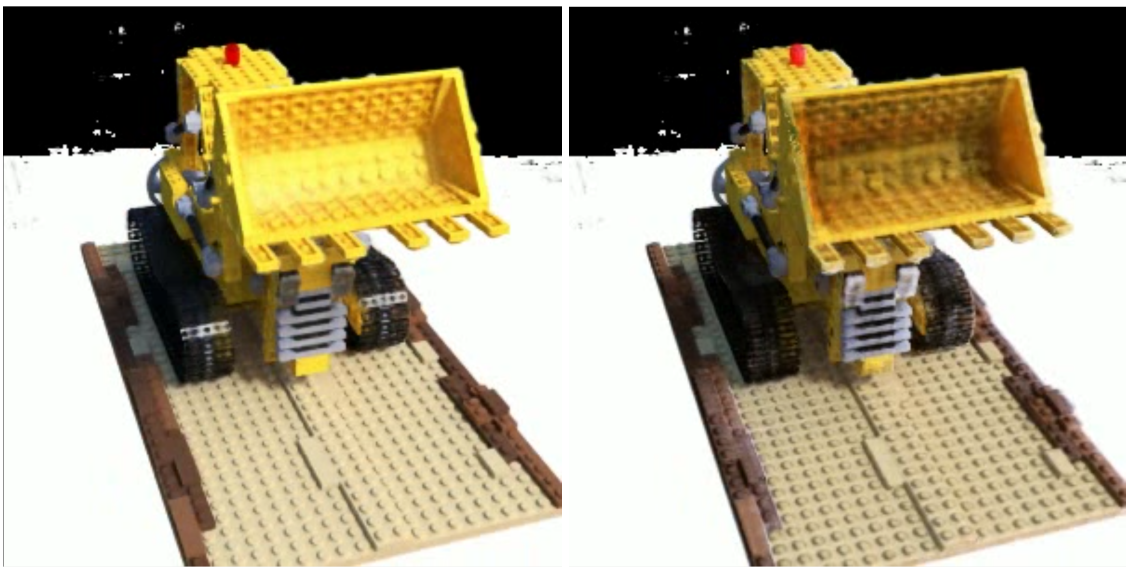
Task 4

On the following page, the renders without constant viewing direction are shown on the left, while the render with constant viewing direction is shown on the right.

There are some notable differences between the two. We may first notice the color intensity of the yellow lego bricks, being much brighter on the left screenshots without constant viewing direction.
In the first set of renders, we see a reflection on the threads of the lego machine in the left screenshot, but this is not present in the right screenshot, which was rendered with a constant viewing direction.
When rendering an object into 3D, we probably want to avoid reflections such as these, as they entail that the object has been captured in a room with such-and-such lighting. Ideally, we choose this ourselves, which is what the constant viewing direction somewhat represents.
Looking at the third and last set of screenshots, it is clear that the constant viewing direction also impacts the render quality negatively from some angles, as the bright areas we see there are clearly mis-representative of the real object.

```python
def prepare_viewdirs_chunks(points, rays_d, encoding_function, chunksize=2**15, constant_viewdir=None):
    """
    Encode and chunkify viewdirs to prepare for NeRF model.
    """
    # Prepare the viewdirs
    if constant_viewdir is not None:
        # TASK 4: Render with a constant viewdir. HINT: Use torch.ones_like(...) and constant_viewdir
        viewdirs = torch.ones_like(rays_d) * constant_viewdir
    else :
        viewdirs = rays_d / torch.norm(rays_d, dim=-1, keepdim=True)
    viewdirs = viewdirs[:, None, ...].expand(points.shape).reshape((-1, 3))
    viewdirs = encoding_function(viewdirs)

    viewdirs = get_chunks(viewdirs, chunksize=chunksize)
    return viewdirs
```



Task 5

First training performed on the chair is shown below, render taken at iteration 9999:

And the ship, also at iteration 9999:

```
rgb_predicted = outputs['rgb_map']
# TASK 5: training loop
#HINT: MSE betweein rgb_predicted and target_img
loss = torch.nn.MSELoss()(rgb_predicted, target_img)
#HINT: zero gradient
optimizer.zero_grad()
#HINT: loss backward
loss.backward()
#HINT optimizer step
optimizer.step()
```