Imports

In [ ]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.cuda.amp import GradScaler, autocast
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from pathlib import Path
import cv2 as cv
from PIL import Image
import segmentation_models_pytorch as smp
from sklearn.model_selection import train_test_split
import albumentations as A
from albumentations.pytorch import ToTensorV2
from torchmetrics.image import StructuralSimilarityIndexMeasure as SSIM
from torchmetrics.regression import MeanSquaredError as MSE
from torchmetrics.collections import MetricCollection
import gc
from torchvision.transforms import Normalize
```

In [ ]:
```
torch.cuda.device_count()
```

Out[ ]:
```
2
```

In [ ]:
```
for i in range(torch.cuda.device_count()):
    print(torch.cuda.get_device_name(i))
```

```
NVIDIA GeForce RTX 2080 Ti
NVIDIA GeForce RTX 2080 Ti
```

In [ ]:
```
# device =
# torch.device('cuda:0'), torch.device('cuda:1'), torch.device('cuda:3')
```

Data & Visualization

Note: brighter the area, farther it is in the image, brighter => more depth

In [ ]:
```
train_csv = Path('./nyu-depth-v2/nyu_data/data/nyu2_train.csv')
train_ims_path = Path('./nyu-depth-v2/nyu_data/data/nyu2_train')
base_path = Path('./nyu-depth-v2/nyu_data')
```

In [ ]:
```
df = pd.read_csv(train_csv,header=None)
df[0] = df[0].map(lambda x:base_path/x)
df[1] = df[1].map(lambda x:base_path/x)
df.head()
```

Out[ ]:

	0	1
0	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...
1	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...
2	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...
3	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...
4	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...	nyu-depth-v2\nyu_data\data\nyu2_train\living_r...

In [ ]:
```
train_df, val_df = train_test_split(df,test_size=0.1,shuffle=True)
val_df, test_df = train_test_split(val_df, test_size=0.1,shuffle=True)
train_df.reset_index(drop=True,inplace=True)
val_df.reset_index(drop=True,inplace=True)
test_df.reset_index(drop=True,inplace=True)
len(train_df),len(val_df), len(test_df)
```

Out[ ]:
```
(45619, 4562, 507)
```

In [ ]:
```
def colored_depthmap(depth, d_min=None, d_max=None,cmap=plt.cm.inferno):
    if d_min is None:
        d_min = np.min(depth)
    if d_max is None:
        d_max = np.max(depth)
    depth_relative = (depth - d_min) / (d_max - d_min)
    return 255 * cmap(depth_relative)[:,:,:3] # H, W, C

def merge_into_row(input, depth_target):
    input = np.array(input)
    depth_target = np.squeeze(np.array(depth_target))

    d_min = np.min(depth_target)
    d_max = np.max(depth_target)
    depth_target_col = colored_depthmap(depth_target, d_min, d_max)
    img_merge = np.hstack([input, depth_target_col])

    return img_merge

plt.figure(figsize=(15, 6))
for i, idx in enumerate(np.random.randint(0,len(df),(16,))):
    ax = plt.subplot(4, 4, i + 1)
    image = Image.open(df.iloc[idx,0]).convert('RGB')
    mask = Image.open(df.iloc[idx,1]).convert('L')
    image_viz = merge_into_row(
        image,mask
    )
    plt.imshow(image_viz.astype("uint8"))
    plt.axis("off")
```

Dataset

In [ ]:
```
sample_tfms = [
    A.HorizontalFlip(),
    A.GaussNoise(p=0.2),
    A.OneOf([
        A.MotionBlur(p=.3),
        A.MedianBlur(blur_limit=3, p=0.3),
        A.Blur(blur_limit=3, p=0.5),
    ], p=0.3),
    A.RGBShift(),
    A.RandomBrightnessContrast(),
    A.RandomResizedCrop(384,384),
    A.ColorJitter(),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.3, rotate_limit=45, p=0.5),
    A.HueSaturationValue(p=0.3),
]
train_tfms = A.Compose([
    *sample_tfms,
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
])
valid_tfms = A.Compose([
    A.Resize(224,224),
    A.Normalize(always_apply=True),
    ToTensorV2()
])
```

In [ ]:
```
class Dataset:
    def __init__(self,df,tfms):
        self.df = df
        self.tfms=tfms
    def open_im(self,p,gray=False):
        im = cv.imread(str(p))
        im = cv.cvtColor(im,cv.COLOR_BGR2GRAY if gray else cv.COLOR_BGR2RGB)
        return im

    def __len__(self,):
        return len(self.df)

    def __getitem__(self,idx):
        s = self.df.iloc[idx,:]
        im, dp = s[0],s[1]
        im, dp = self.open_im(im), self.open_im(dp,True)
```

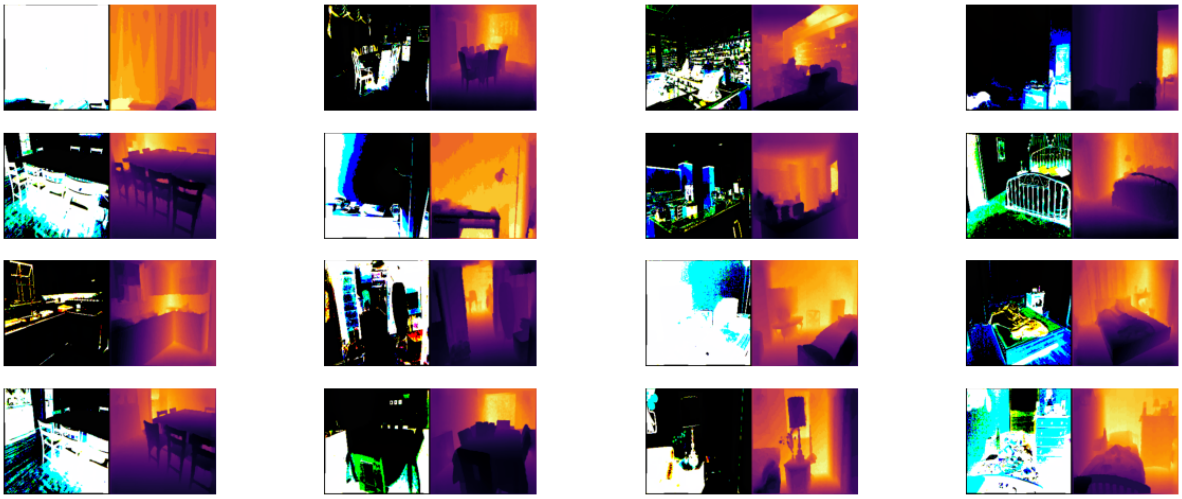
```
        augs = self.tfms(image=im,mask=dp)
        im, dp = augs['image'], augs['mask'] / 255.
        return im, dp.unsqueeze(0)
```

In [ ]:
```
train_ds = Dataset(train_df,train_tfms)
val_ds = Dataset(val_df,valid_tfms)
test_ds = Dataset(test_df, valid_tfms)
len(train_ds), len(val_ds), len(test_ds)
```

Out[ ]:
```
(45619, 4562, 507)
```

Transforms Visualization

In [ ]:
```
randoms = [test_ds[idx] for idx in range(16)]
plt.figure(figsize=(15, 6))
for i, (img,mask) in enumerate(randoms):
    ax = plt.subplot(4, 4, i + 1)
    img, mask = img.permute(1,2,0).numpy(), mask.permute(1,2,0).numpy()
    mask = colored_depthmap(np.squeeze(mask))
    image_viz = np.hstack([img,mask])
    plt.imshow(image_viz.astype("uint8"))
    plt.axis("off")
```



In [ ]:
```
train_dl = torch.utils.data.DataLoader(train_ds,shuffle=True,batch_size=64)
val_dl = torch.utils.data.DataLoader(val_ds,shuffle=False,batch_size=64)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle=True, batch_size=4)
len(train_dl), len(val_dl), len(test_dl)
```

Out[ ]:
```
(713, 72, 127)
```

Model

In [ ]:
```
class UNet(nn.Module):
    def __init__(self,):
        super().__init__()
        self.model = smp.UnetPlusPlus(
            encoder_name='resnext50_32x4d',
            in_channels=3,
            classes=1
        )

    def trainable_encoder(self,trainable=True):
```

```
        for p in self.model.encoder.parameters():
            p.requires_grad = trainable

    def forward(self,x):
        return self.model(x)

    def _num_params(self,):
        return sum([p.numel() for p in self.model.parameters() if p.requires_grad])
```

Training

Metrics:

Structural Similarity Index (SSIM)

The structural similarity index (SSIM) is a perceptual metric that takes into account the structural similarity of the two images. SSIM is calculated by comparing the local patterns of the two images, taking into account the luminance, contrast, and structure of the images.

The formula for SSIM is:

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μx and μy are the means of the two images, σx2 and σy2 are the variances of the two images, and σxy is the covariance of the two images. c1 and c2 are constants that are used to stabilize the calculation of SSIM.

SSIM is more robust to noise and small changes than MSE, but it is also more computationally expensive to calculate.

In general, SSIM is preferred over MSE for image quality assessment because it provides a more accurate measure of how humans perceive the similarity of two images. However, MSE is still a useful metric, especially when speed is important.

In [ ]:
```
class UnNormalize(Normalize):
    def __init__(self,*args,**kwargs):
        mean=(0.485, 0.456, 0.406)
        std=(0.229, 0.224, 0.225)
        new_mean = [-m/s for m,s in zip(mean,std)]
        new_std = [1/s for s in std]
        super().__init__(new_mean, new_std, *args, **kwargs)

@torch.no_grad()
def plot_vals(imgs, preds, targets,n=4,figsize=(6,2),title=''):
    plt.figure(figsize=figsize,dpi=150)
    r = 2 if n == 4 else 8
    c = 2
    for i,idx in enumerate(np.random.randint(0,imgs.size(0),(n,))):
        ax = plt.subplot(r,c,i + 1)
        img,pred,gt = imgs[idx], preds[idx], targets[idx]
        img = UnNormalize()(img)*255.
        img,pred,gt = img.permute(1,2,0).numpy(), pred.permute(1,2,0).numpy(), gt.pe
        pred = colored_depthmap(np.squeeze(pred))
        gt = colored_depthmap(np.squeeze(gt))
        image_viz = np.hstack([img,pred,gt])
```

```
        plt.imshow(image_viz.astype("uint8"))
        plt.axis("off")
    title = f'{title}\nimage/target/prediction' if len(title)!=0 else 'image/target
    plt.suptitle(title)
    plt.show()
```

In [ ]:
```
epochs = 5
freeze_epochs = 2
lr = 1e-3

device = torch.device('cuda:1')

metrics = MetricCollection([
    SSIM(data_range=(0,1)),
    MSE()
]).to(device)
train_metrics = metrics.clone()
val_metrics = metrics.clone()

logs = pd.DataFrame()
logs[['loss_train','loss_val','ssim_train','ssim_val','mse_train','mse_val']]=None

model = UNet().to(device)
model.trainable_encoder(trainable=False)

loss_fn = nn.MSELoss()
optim = torch.optim.AdamW(model.parameters(),lr=lr / 25.,weight_decay=0.02)
sched = torch.optim.lr_scheduler.OneCycleLR(optim,max_lr=lr,epochs=epochs,steps_per_
```

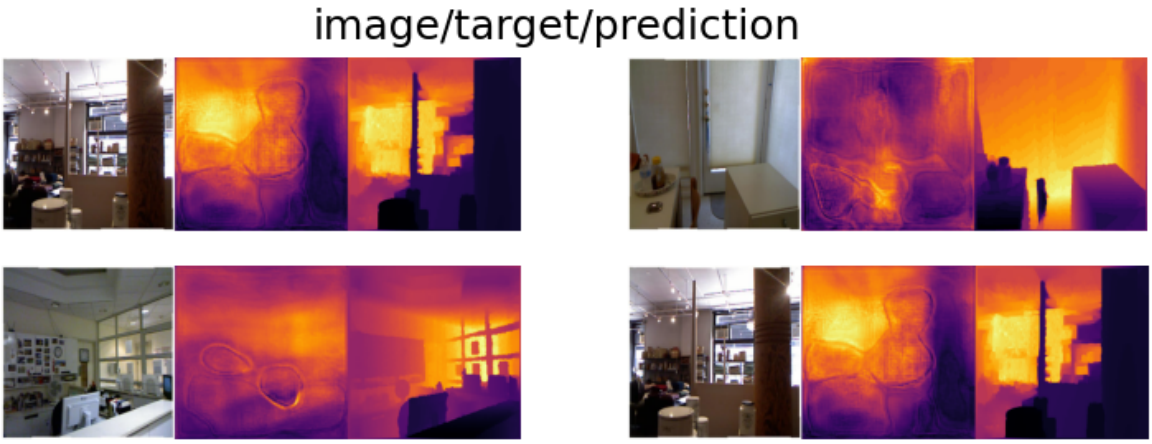
```
scaler = GradScaler()
```

In [ ]:
```
best_ssim = -1e9
best_epoch = -1

print('training decoder only')

for epoch in tqdm(range(epochs)):

    model.train()

    if epoch == freeze_epochs:
        model.trainable_encoder(trainable=True)
        print('training encoder and decoder both')

    running_loss = 0.
    train_prog = tqdm(train_dl,total=len(train_dl))

    for img, mask in train_prog:

        with autocast():
            img, mask = img.to(device), mask.to(device)
            preds = model(img)

            loss = loss_fn(preds,mask)
            scaler.scale(loss).backward()
            scaler.unscale_(optim)
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0, norm_type=2)
            scaler.step(optim)
            scaler.update()
            sched.step()
            optim.zero_grad()

            running_loss += loss.item()
```

```
            train_prog.set_description(f'loss: {loss.item():.3f}')
            train_metrics(preds,mask)

        del img, mask, preds, loss

    m = train_metrics.compute()
    _ssim,_mse = m['StructuralSimilarityIndexMeasure'].cpu().item(), m['MeanSquaredE
    logs.loc[epoch,['loss_train','ssim_train','mse_train']] = (running_loss/len(trai
    train_metrics.reset()
    model.eval()

    with torch.no_grad():

        running_loss = 0.

        val_prog = tqdm(val_dl,total=len(val_dl))
        for img, mask in val_prog:

            with autocast():
                img, mask = img.to(device), mask.to(device)
                preds = model(img)
                loss = loss_fn(preds,mask)
                running_loss += loss.item()
                val_prog.set_description(f'loss: {loss.item():.3f}')

                val_metrics(preds, mask)

            del img, mask, preds, loss

        m = val_metrics.compute()
        _ssim,_mse = m['StructuralSimilarityIndexMeasure'].cpu().item(), m['MeanSqua
        logs.loc[epoch,['loss_val','ssim_val','mse_val']] = (running_loss/len(val_dl
        val_metrics.reset()

    if _ssim > best_ssim:
        best_ssim = _ssim
        best_epoch = epoch
        sd = model.state_dict()
        torch.save(sd, Path('./model-weights/nyu-v2-depth-resnext50_32x4d-unetpluspl

    print(f"\n\n{logs.tail(1)}\n\n")

    with torch.no_grad():
        with autocast():
            img, mask = next(iter(test_dl))
            img, mask = img.to(device), mask.to(device)
            preds = model(img)
            plot_vals(
                img.cpu(),
                preds.cpu(),
                mask.cpu()
            )

    gc.collect()
    torch.cuda.empty_cache()
```

```
training decoder only
  0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/713 [00:00<?, ?it/s]
  0%|          | 0/72 [00:00<?, ?it/s]
```

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
0	0.03118	0.008758	0.578338	0.769319	0.031186	0.008767




```
  0%|          | 0/713 [00:00<?, ?it/s]
  0%|          | 0/72 [00:00<?, ?it/s]
```

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
1	0.010111	0.006337	0.853832	0.863473	0.010111	0.006352


```
training encoder and decoder both
  0%|          | 0/713 [00:00<?, ?it/s]
  0%|          | 0/72 [00:00<?, ?it/s]
```

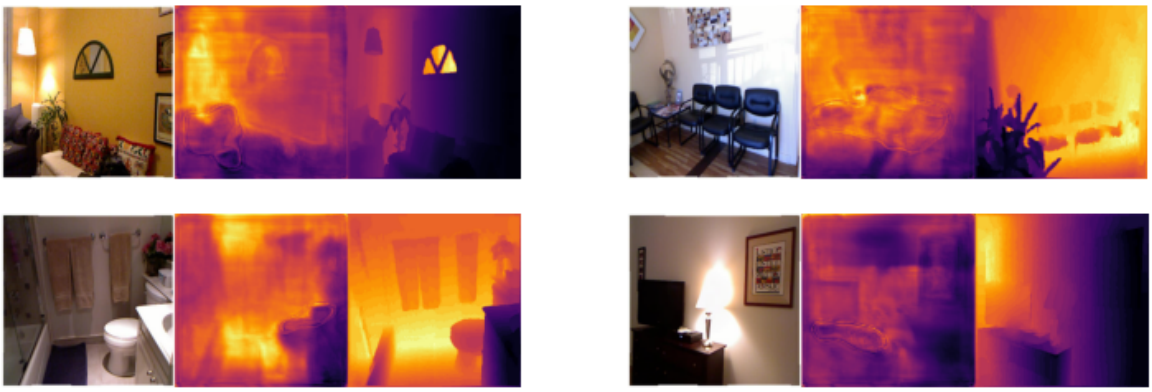
	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
2	0.010468	0.005191	0.875593	0.891569	0.010469	0.005197

image/target/prediction

```
  0%|          | 0/713 [00:00<?, ?it/s]
  0%|          | 0/72 [00:00<?, ?it/s]
```

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
3	0.006773	0.003099	0.898602	0.90524	0.006773	0.003106

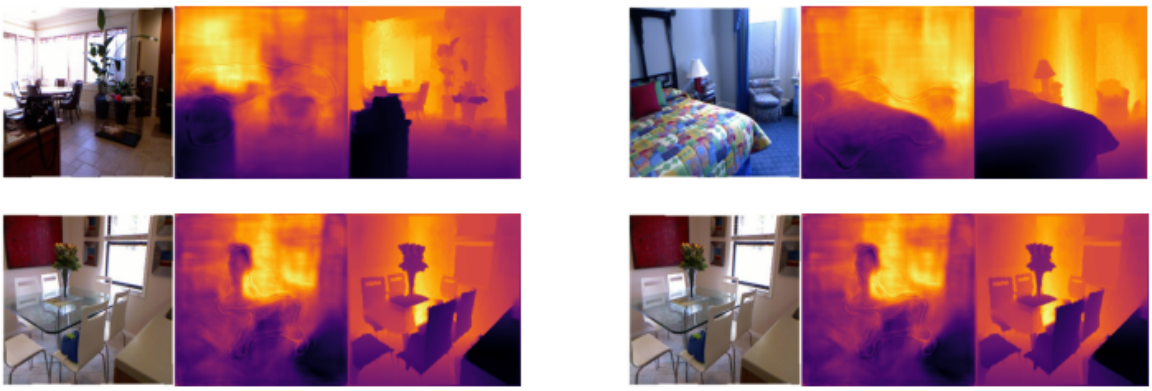
image/target/prediction



```
  0%|          | 0/713 [00:00<?, ?it/s]
  0%|          | 0/72 [00:00<?, ?it/s]
```

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
4	0.004926	0.002794	0.909537	0.908052	0.004927	0.002798

image/target/prediction



Predictions

In [ ]:
```
best_epoch
```

Out[ ]:
```
4
```

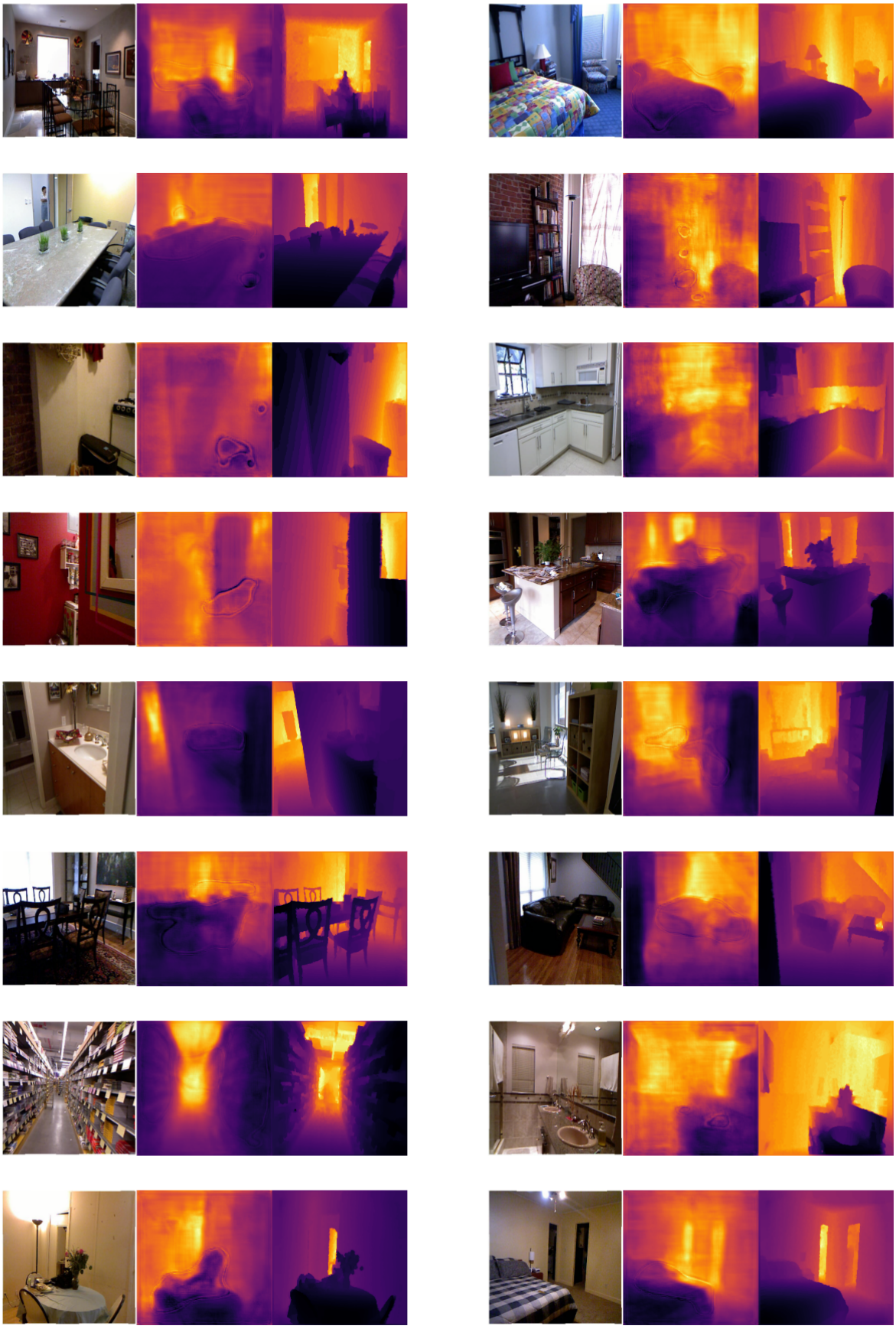
In [ ]:
```
best_sd = torch.load(Path('./model-weights/nyu-v2-depth-resnext50_32x4d-unetplusplus
model.load_state_dict(best_sd)
```

Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
all_imgs, all_preds, all_targets = [], [], []
with torch.no_grad():
    with autocast():
        for img, mask in tqdm(test_dl,total=len(test_dl)):
            img, mask = img.to(device), mask.to(device)
            preds = model(img)
            all_imgs.append(img)
            all_preds.append(preds)
            all_targets.append(mask)

test_metrics = metrics.clone()
test_metrics(
    torch.vstack(all_preds),
    torch.vstack(all_targets)
)
m = test_metrics.compute()
title = f"SSIM: {m['StructuralSimilarityIndexMeasure'].cpu().item():.3f} MSE: {m['Me
plot_vals(
    torch.vstack(all_imgs).cpu(),
    torch.vstack(all_preds).cpu(),
    torch.vstack(all_targets).cpu(),
    n=16,
    figsize=(10,15),
    title=title
)
```

```
  0%|          | 0/127 [00:00<?, ?it/s]
```

SSIM: 0.912 MSE: 0.003
image/target/prediction



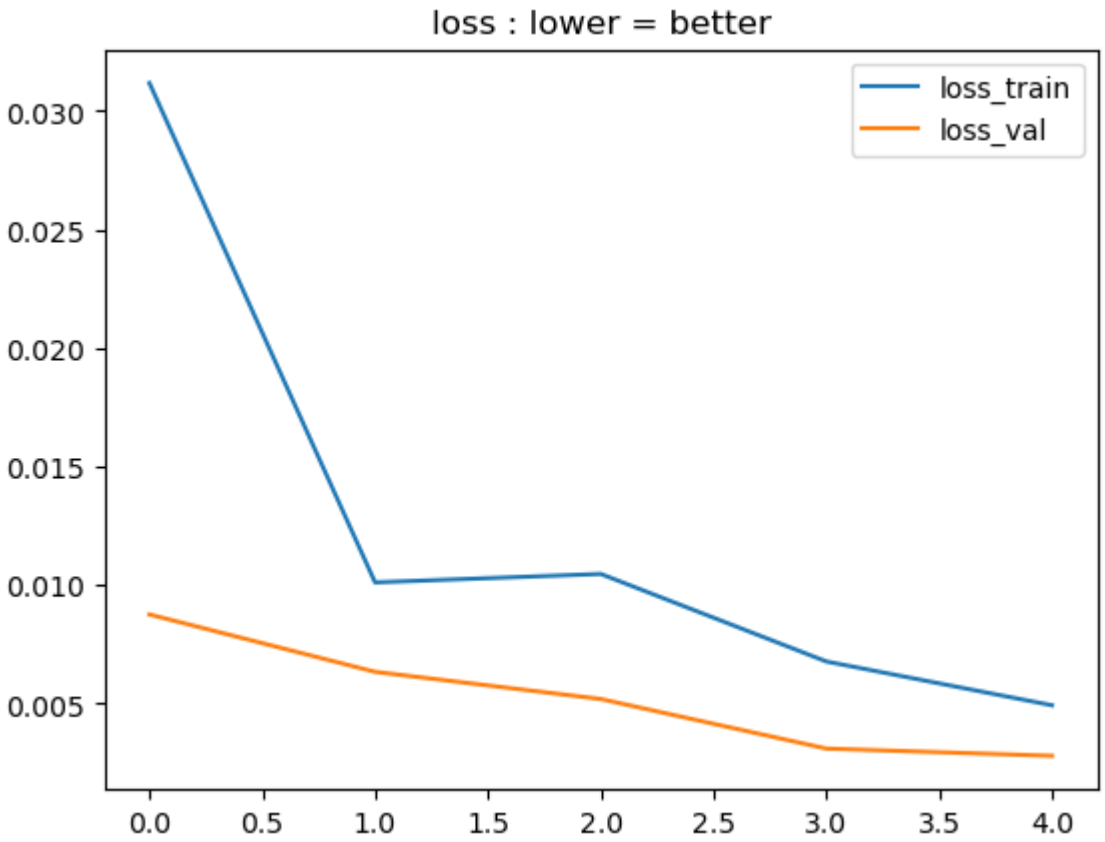
Results

In [ ]:
```
logs
```

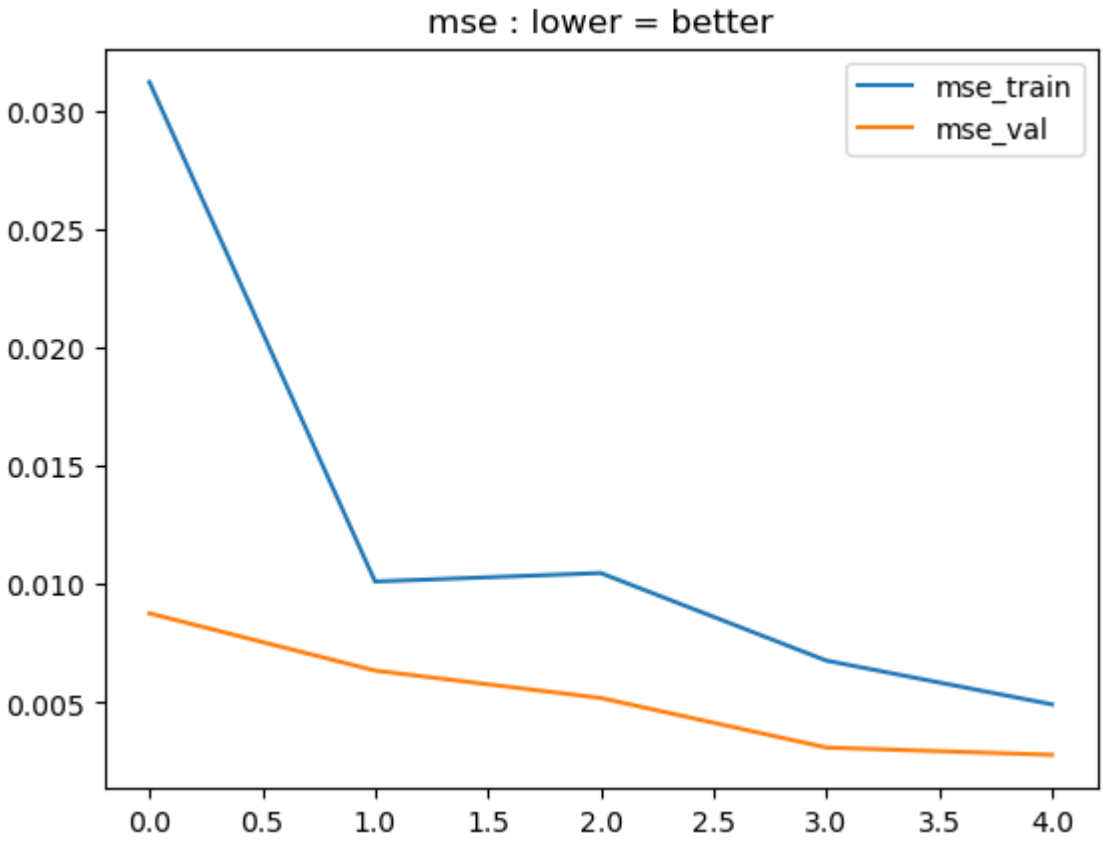
Out[ ]:

	loss_train	loss_val	ssim_train	ssim_val	mse_train	mse_val
0	0.03118	0.008758	0.578338	0.769319	0.031186	0.008767
1	0.010111	0.006337	0.853832	0.863473	0.010111	0.006352
2	0.010468	0.005191	0.875593	0.891569	0.010469	0.005197
3	0.006773	0.003099	0.898602	0.90524	0.006773	0.003106
4	0.004926	0.002794	0.909537	0.908052	0.004927	0.002798

In [ ]:
```
logs['loss_train'].plot()
logs['loss_val'].plot()
plt.title('loss : lower = better')
plt.legend()
plt.show()
```



In [ ]:
```
logs['mse_train'].plot()
logs['mse_val'].plot()
plt.title('mse : lower = better')
plt.legend()
plt.show()
```



In [ ]:
```
logs['ssim_train'].plot()
logs['ssim_val'].plot()
plt.title('ssim : higher = better')
plt.legend()
plt.show()
```