In 5
```
%cd /content/drive/MyDrive/DepthSense
!ls
```

```
/content/drive/MyDrive/DepthSense
data     loss    notes     online_sampling  ReOWeb_V1          scratchpad.ipynb  utils
LICENSE  models  notes.md  README.md        ReOWeb_V1.tar.gz   scripts
```

In 6
```
!ls -lh
```

```
-rw------- 1 root root 2.3K Dec  2 18:52 notes.md
drwx------ 2 root root 4.0K Dec  4 19:40 online_sampling
-rw------- 1 root root 1.1K Dec  2 20:42 README.md
drwx------ 2 root root 4.0K Dec  5 08:43 ReOWeb_V1
-r-------- 1 root root 277M Feb 24  2019 ReOWeb_V1.tar.gz
-rw------- 1 root root 200K Dec  4 19:13 scratchpad.ipynb
drwx------ 2 root root 4.0K Dec  4 19:40 scripts
drwx------ 2 root root 4.0K Dec  4 19:40 utils
```

In 7
```
# !tar -xvf ReOWeb_V1.tar.gz
```

In 8
```
!ls ReOWeb_V1/Imgs/ | wc -l
```

```
3600
```

In 9
```
!ls ReOWeb_V1/RDs/ | wc -l
```

```
3600
```

In 10
```
import matplotlib.pyplot as plt
import torch
from torchvision.io import read_image
from torchvision import transforms
import torch.nn as nn
import torch.nn.functional as F


from torch.utils.data import DataLoader
```

In 11
```
from data.loaders.DataLoader import ReOWebDataset , Rescale , RandomCrop
normal_dataset = ReOWebDataset(root_dir="ReOWeb_V1",transform=transforms.Compose([
    Rescale((256,256)),
]))
batcher = DataLoader(normal_dataset,batch_size=1,shuffle=True)
```

In 14
```
from loss.InverseDepthLoss import InverseDepthSmoothnessLoss as IDSL
```

In 15
```
idsl = IDSL()
sample0["mono"] =  sample0["mono"].to(torch.float)
sample0["heat"] =  sample0["heat"].to(torch.float)

sample1["mono"] = sample1["mono"].to(torch.float)
sample1["heat"] = sample1["heat"].to(torch.float)
print(idsl(sample0["heat"],sample0["mono"]))
print(idsl(sample0["heat"],sample1["mono"]))
```

```
tensor(0.0743)
tensor(0.1085)
```

In 16
```
## Modelling
```

In 17
```
import torch
import torch.nn as nn
import torchvision.models as models

class ResNetWithFeatureFusion(nn.Module):
    def __init__(self):
        super(ResNetWithFeatureFusion, self).__init__()

        # Load the pretrained ResNet50 model
        self.resnet50 = models.resnet50(pretrained=True)

        # Extract features from intermediate layers
        self.layer1_features = nn.Sequential(
            self.resnet50.conv1,
            self.resnet50.bn1,
            self.resnet50.relu,
            self.resnet50.maxpool,
            self.resnet50.layer1
        )

        self.layer2_features = self.resnet50.layer2
        self.layer3_features = self.resnet50.layer3

        # Additional layers for feature fusion
        self.fusion_conv = nn.Conv2d(in_channels=1024, out_channels=3, kernel_size=1)
        self.fusion_bn = nn.BatchNorm2d(3)
        self.fusion_relu = nn.ReLU(inplace=True)

    def __call__(self, x):
        # Backbone ResNet
        x = self.layer1_features(x)
        x = self.layer2_features(x)
        x = self.layer3_features(x)

        # Feature Fusion
        fused_features = self.fusion_conv(x)
        fused_features = self.fusion_bn(fused_features)
        fused_features = self.fusion_relu(fused_features)
        fused_features = F.interpolate(fused_features, size=(256, 256), mode='bilinear', align_corners=False)

        return fused_features

# Example usage:
model = ResNetWithFeatureFusion()
```

```
  0.13 and may be removed in the future, please use 'weights' instead.
    warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
    warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|[redacted]| 97.8M/97.8M [00:00<00:00, 126MB/s]
```

In _
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms



device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_epochs = 10
learning_rate = 0.001
batch_size = 32

dataset = normal_dataset
dataloader =  DataLoader(dataset,batch_size=batch_size,shuffle=True)

model = ResNetWithFeatureFusion().to(device)

criterion = IDSL()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0

    for batch in dataloader:
        inputs = batch['mono'].to(device)
        targets = batch['heat'].to(device)
        inputs = inputs.float()

        # flush the gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)

        outputs = outputs.float()
        targets = targets.float()
        # Calculate the loss
        # print(targets.shape , outputs.shape)
        loss = idsl(targets,outputs)
        print("loss -=",loss)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    # Print the average loss for the epoch
    average_loss = total_loss / len(dataloader)
    print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {average_loss:.4f}")

torch.save(model.state_dict(), 'resnet_with_feature_fusion.pth')
```

```
loss -= tensor(2.2738, grad_fn=<AddBackward0>)
loss -= tensor(2.1302, grad_fn=<AddBackward0>)
loss -= tensor(2.0845, grad_fn=<AddBackward0>)
loss -= tensor(2.2161, grad_fn=<AddBackward0>)
loss -= tensor(2.2163, grad_fn=<AddBackward0>)
loss -= tensor(2.1896, grad_fn=<AddBackward0>)
loss -= tensor(2.0759, grad_fn=<AddBackward0>)
loss -= tensor(2.2854, grad_fn=<AddBackward0>)
```