Boundary Refinement of Remote Sensing Images

PROJECT GUIDE:

DR. M. SUBRAMANIAM

SAMREEN SULTHANA

Agenda:

- Problem Statement
- Introduction
- Working principle
- Implementation
- Conclusion

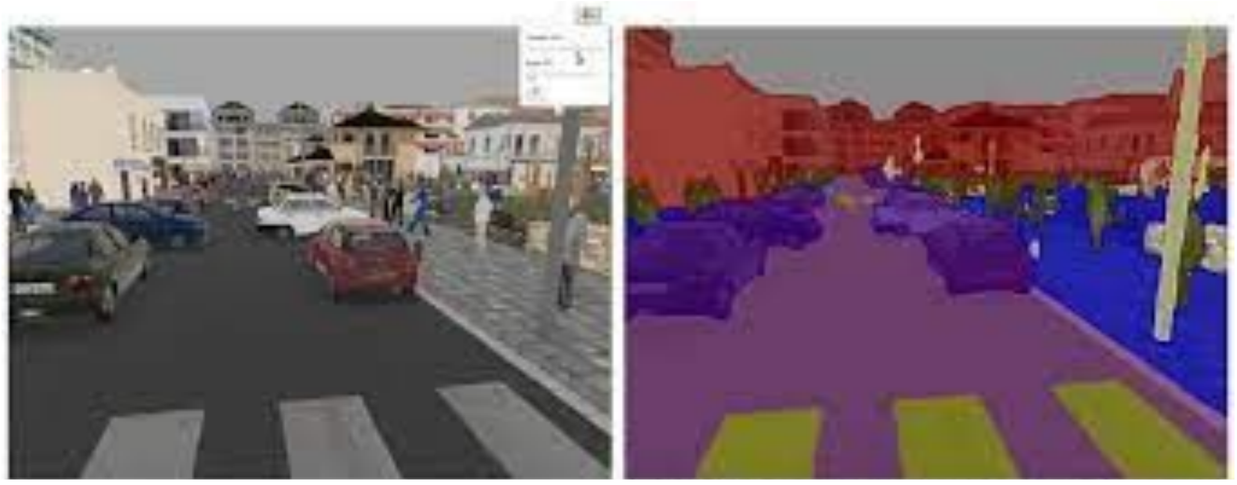
Problem Statement

Remote sensing images helps in many parts of human development from Urban City planning to Disaster Management.As part of remote sensing images we need to identify the object’s boundary carefully.Objects like buildings are usually of certain regular geometrical shape like rectangle or a combination of multiple geometrical shapes.So, the absolute annotation of boundary lines is extremely important.We are developing a working model which refines the boundaries of buildings of Remote Sensing images.

Introduction

Semantic Segmentation

Semantic segmentation is a computer vision technique that involves dividing an image into multiple segments or regions, where each region corresponds to a particular object, feature, or class. The goal of semantic segmentation is to identify and label every pixel in an image with a corresponding class label.



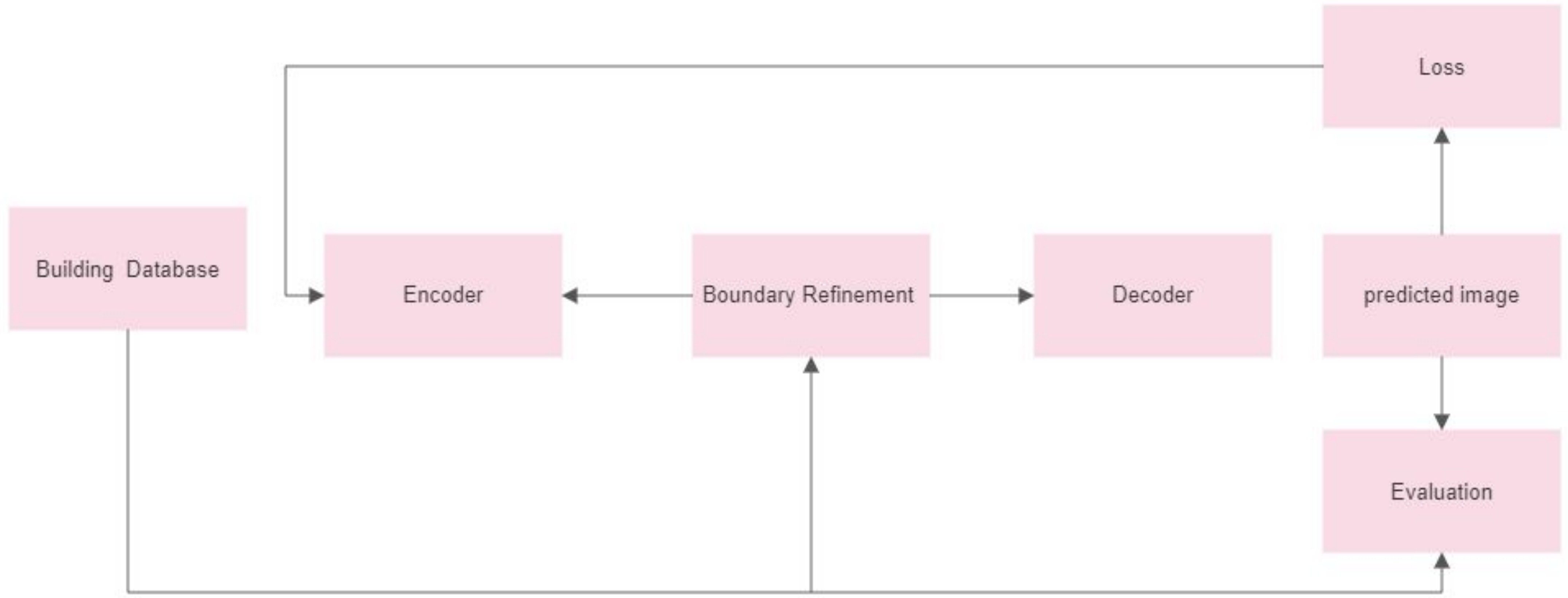
Semantic Segmentation …

- The process of semantic segmentation involves training a machine learning model on a dataset of labeled images. The model learns to identify patterns and features in the images that correspond to specific classes or labels. Once trained, the model can be used to predict the class labels of new images.
- Semantic segmentation is a powerful tool for image analysis, as it allows for detailed and accurate classification of images. It can help to identify patterns and trends in large datasets, making it useful for a wide range of applications.

Boundary Refinement

- Boundary refinement is used in aerial images to improve the accuracy and precision of the boundaries or edges of a specific area or property. In aerial photography, images are captured from a high altitude, and it can be challenging to identify and distinguish the boundaries of individual properties or areas.
- Boundary refinement helps to enhance the quality and clarity of aerial images, making it easier to identify and analyse specific features, such as land use patterns, property boundaries, and environmental changes.

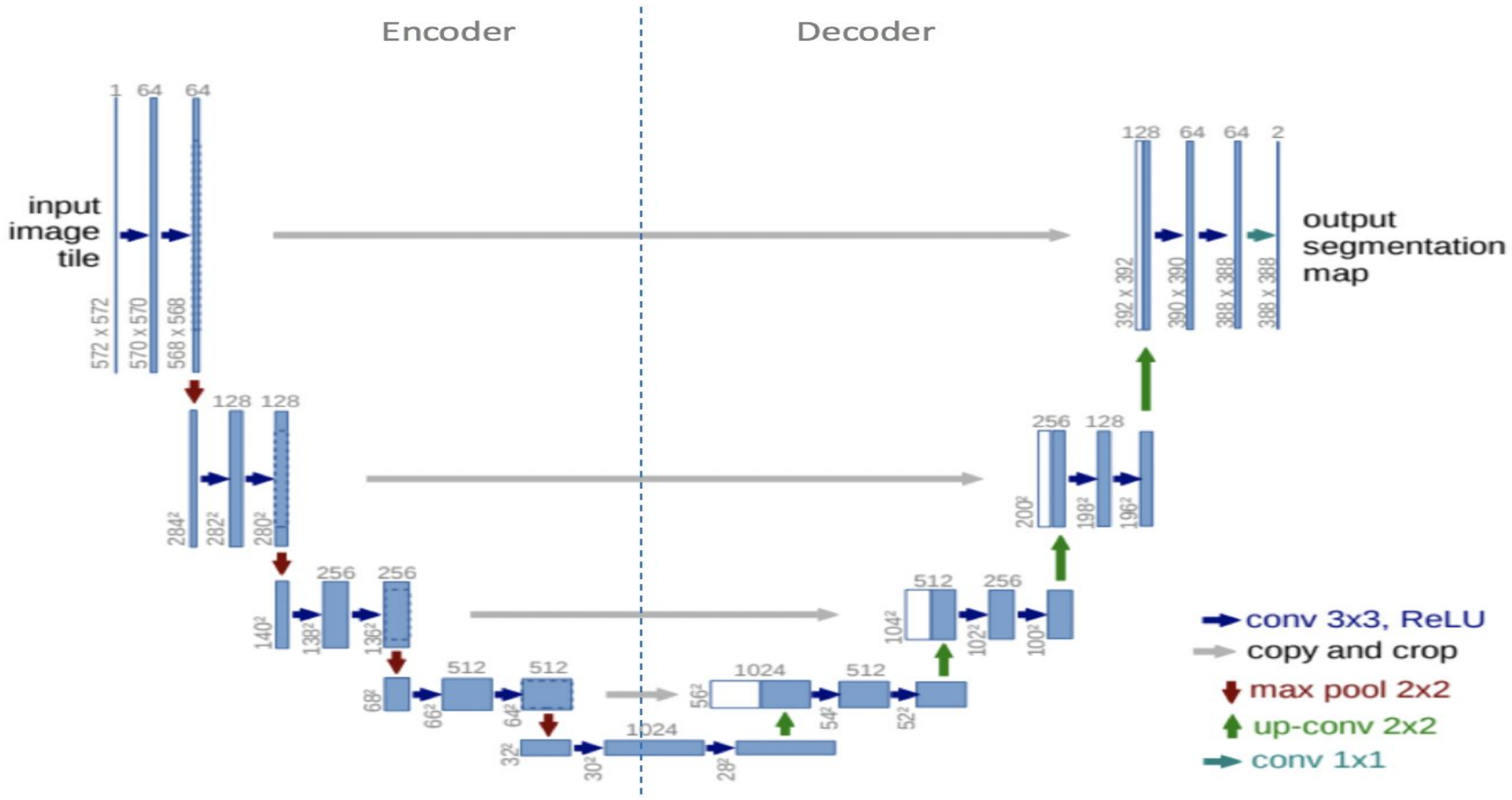
Working Principle:



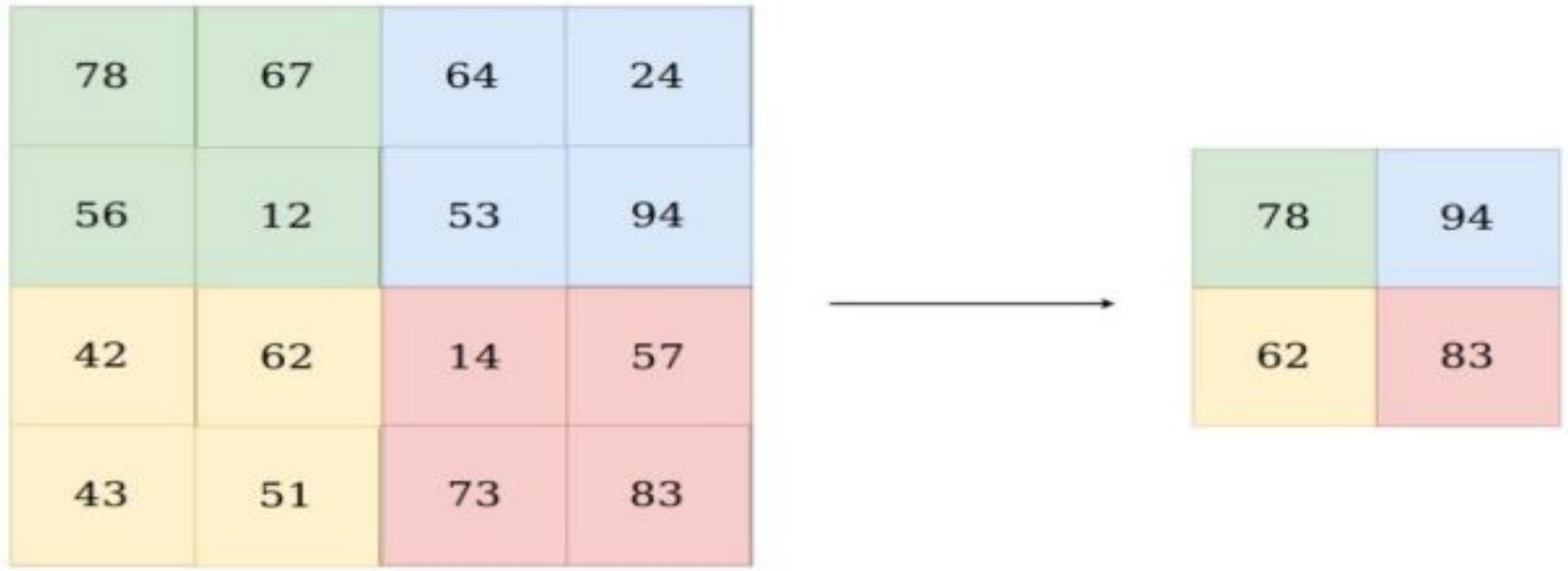
Massachusetts Buildings Dataset

- The Massachusetts Buildings Dataset consists of 151 aerial images of the Boston area, with each of the images being 1500 × 1500 pixels.
- The entire dataset covers roughly 340 square kilometers.
- The data is split into a training set of 137 images, a test set of 10 images and a validation set of 4 images.
- All imagery is rescaled to a resolution of 1 pixel per square meter.

UNet Architecture



- U-Net is an architecture for semantic segmentation. It consists of a contracting path and an expansive path .
- The contracting path follows the architecture of a **convolutional network.** It consists of the repeated application of two 3x3 convolutions , each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation for downsampling.



78	67	64	24
56	12	53	94
42	62	14	57
43	51	73	83

78	94
62	83

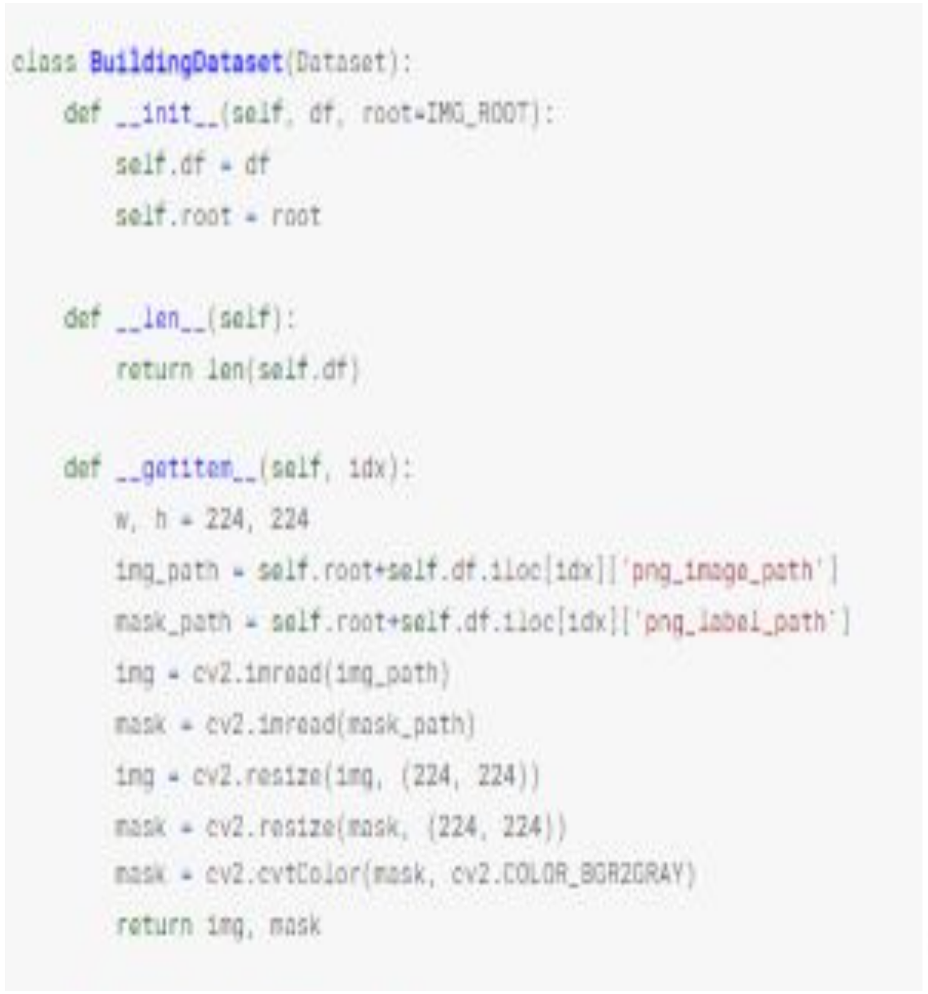
- At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU.
- The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

Cross Entropy loss Function

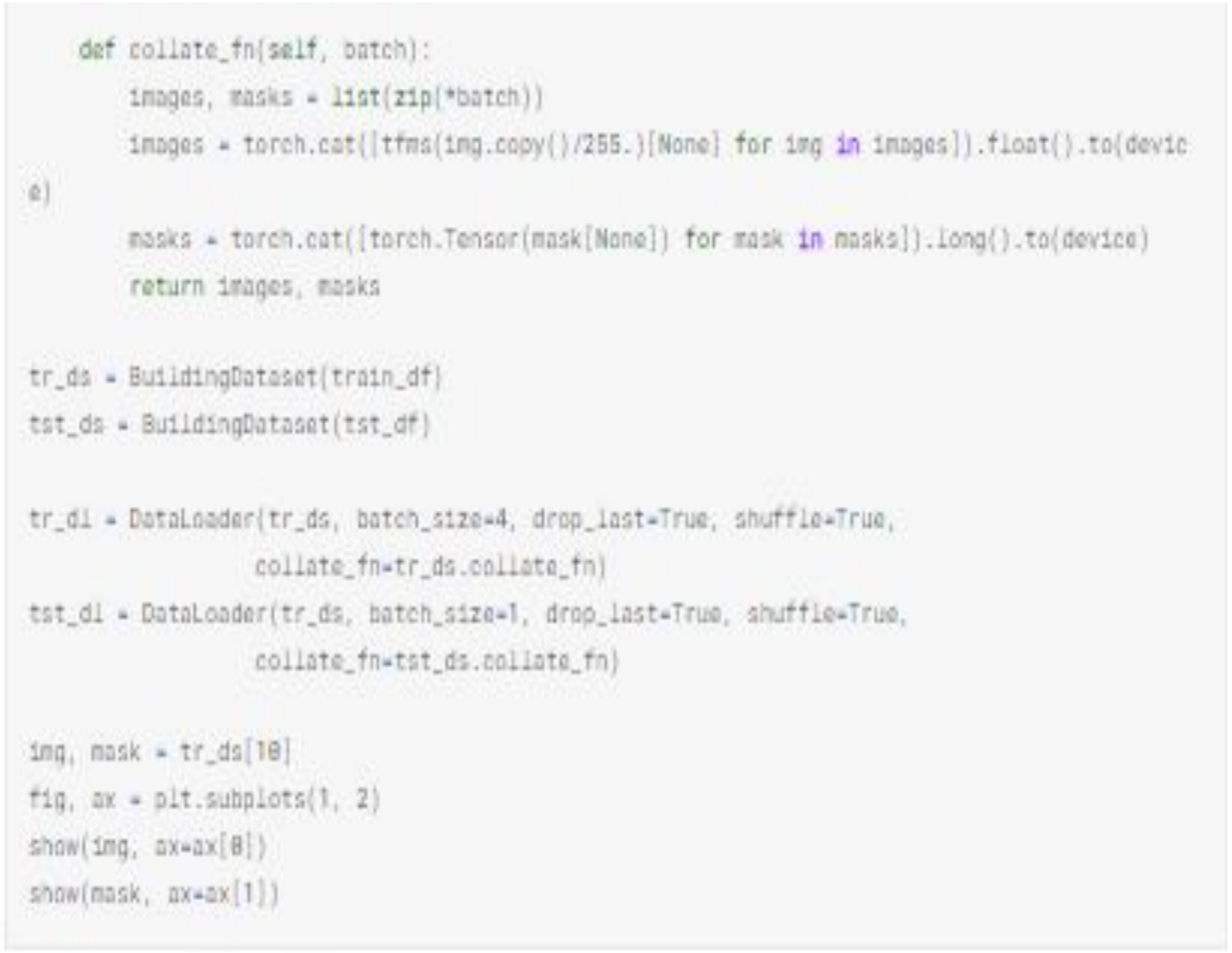
Cross entropy loss (CE) is the most commonly used loss function in dense semantic annotation tasks. It can be described as:

$$CE(p, y) = -\sum y \log(p)$$

Building Dataset

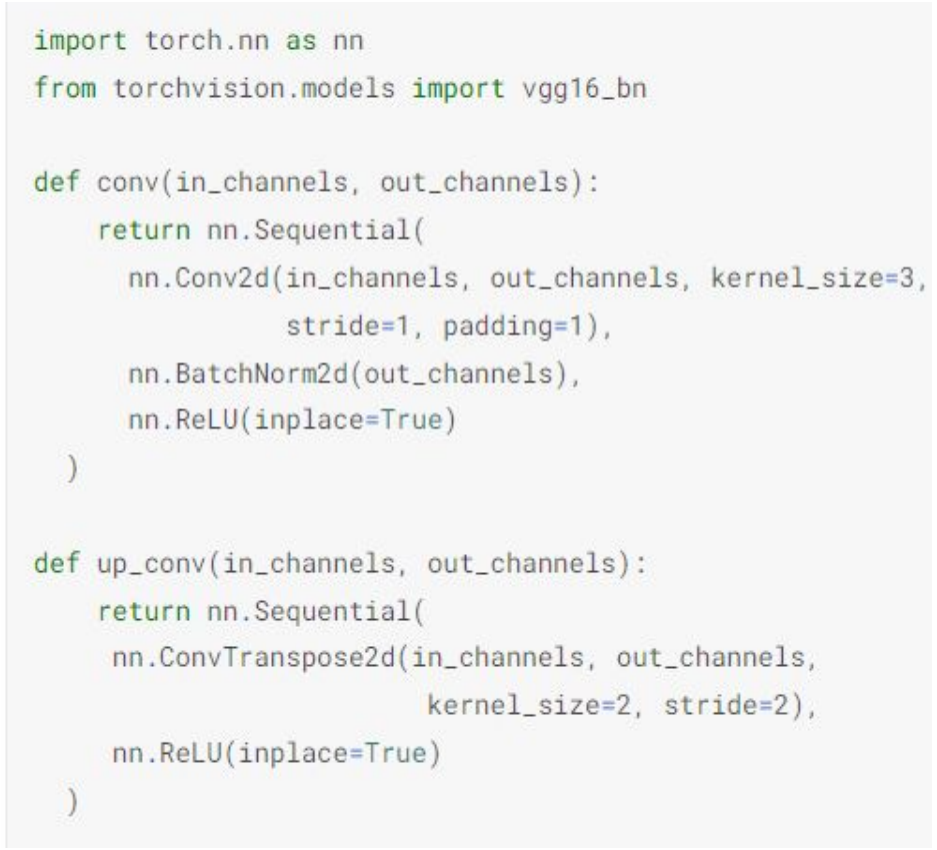


```
class BuildingDataset(Dataset):
    def __init__(self, df, root=IMG_ROOT):
        self.df = df
        self.root = root

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

    def __getitem__(self, idx):
        w, h = 224, 224
        img_path = self.root+self.df.iloc[idx]['png_image_path']
        mask_path = self.root+self.df.iloc[idx]['png_label_path']
        img = cv2.imread(img_path)
        mask = cv2.imread(mask_path)
        img = cv2.resize(img, (224, 224))
        mask = cv2.resize(mask, (224, 224))
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
        return img, mask
```

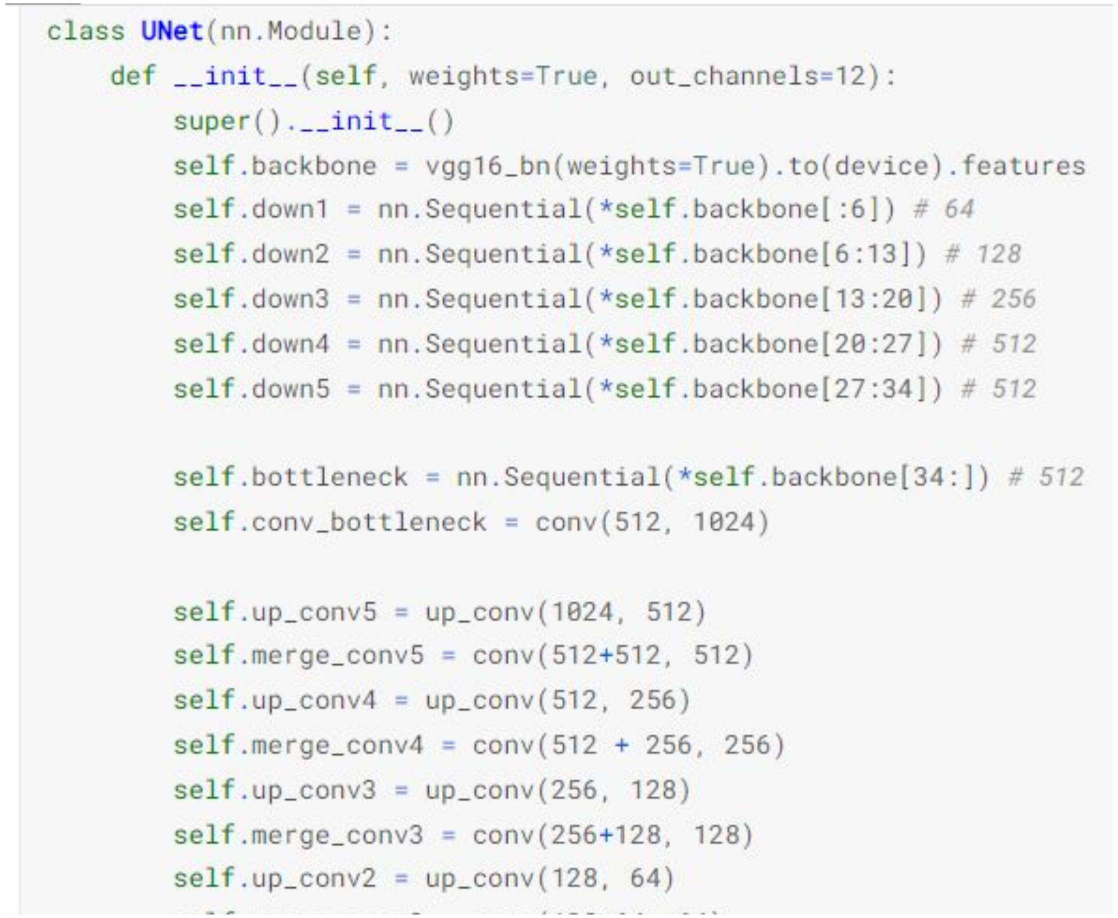


```
    def collate_fn(self, batch):
        images, masks = list(zip(*batch))
        images = torch.cat([tfms(img.copy()/255.)[None] for img in images]).float().to(devic
e)
        masks = torch.cat([torch.Tensor(mask[None]) for mask in masks]).long().to(device)
        return images, masks

tr_ds = BuildingDataset(train_df)
tst_ds = BuildingDataset(tst_df)

tr_dl = DataLoader(tr_ds, batch_size=4, drop_last=True, shuffle=True,
                   collate_fn=tr_ds.collate_fn)
tst_dl = DataLoader(tr_ds, batch_size=1, drop_last=True, shuffle=True,
                   collate_fn=tst_ds.collate_fn)

img, mask = tr_ds[10]
fig, ax = plt.subplots(1, 2)
show(img, ax=ax[0])
show(mask, ax=ax[1])
```

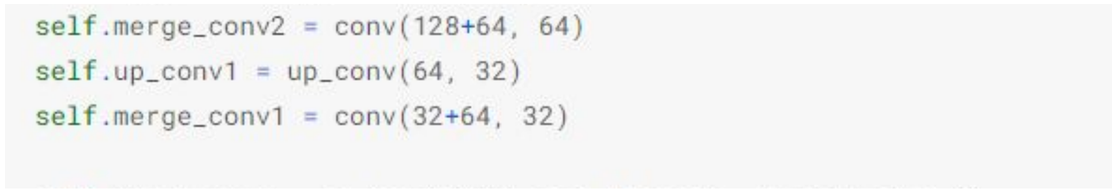
Unet



```python
import torch.nn as nn
from torchvision.models import vgg16_bn

def conv(in_channels, out_channels):
    return nn.Sequential(
      nn.Conv2d(in_channels, out_channels, kernel_size=3,
                stride=1, padding=1),
      nn.BatchNorm2d(out_channels),
      nn.ReLU(inplace=True)
  )

def up_conv(in_channels, out_channels):
    return nn.Sequential(
     nn.ConvTranspose2d(in_channels, out_channels,
                        kernel_size=2, stride=2),
     nn.ReLU(inplace=True)
  )
```



```python
class UNet(nn.Module):
    def __init__(self, weights=True, out_channels=12):
        super().__init__()
        self.backbone = vgg16_bn(weights=True).to(device).features
        self.down1 = nn.Sequential(*self.backbone[:6]) # 64
        self.down2 = nn.Sequential(*self.backbone[6:13]) # 128
        self.down3 = nn.Sequential(*self.backbone[13:20]) # 256
        self.down4 = nn.Sequential(*self.backbone[20:27]) # 512
        self.down5 = nn.Sequential(*self.backbone[27:34]) # 512

        self.bottleneck = nn.Sequential(*self.backbone[34:]) # 512
        self.conv_bottleneck = conv(512, 1024)

        self.up_conv5 = up_conv(1024, 512)
        self.merge_conv5 = conv(512+512, 512)
        self.up_conv4 = up_conv(512, 256)
        self.merge_conv4 = conv(512 + 256, 256)
        self.up_conv3 = up_conv(256, 128)
        self.merge_conv3 = conv(256+128, 128)
        self.up_conv2 = up_conv(128, 64)
```



```python
self.merge_conv2 = conv(128+64, 64)
self.up_conv1 = up_conv(64, 32)
self.merge_conv1 = conv(32+64, 32)
```

Conclusion

Boundary refinement helps to enhance the quality and clarity of aerial images, making it easier to identify and analyse specific features, such as land patterns, property boundaries, and environmental changes.This is especially important in applications such as land surveying, urban planning, real estate, and environmental monitoring, where accurate and precise boundary information is critical for decision-making.

References

[1].Lin Luo, Pengpeng Li ,and Xuesong Yan,” Deep Learning-Based Building Extraction from Remote Sensing Images”, MDPI,2021.

[2] A Boundary Regulated Network for Accurate Roof Segmentation and Outline Extraction

[3] Building segmentation through a gated graph convolutional neural network with deep structured feature embedding

[4] An Improved Boundary-Aware Perceptual Loss for Building Extraction from VHR Images

Thank you