Satellite Image Segmentation By Land Use

By Luan Nguyen, Nick(Chi Yen Yu), Juan, Ethan, Alan

Background

Takes satellite images taken by Sentinel-2 categorize use of land into 10 different categories.

Examples of practical use:

City planning / Real Estate

Environmental Monitoring

Dataset

Satellite images taken by Sentinel-2.

Containing 27,000 images (64x64, 10m definition).

Divided into 10 categories of usage.

Two options: RGB or 12-band.

About this directory

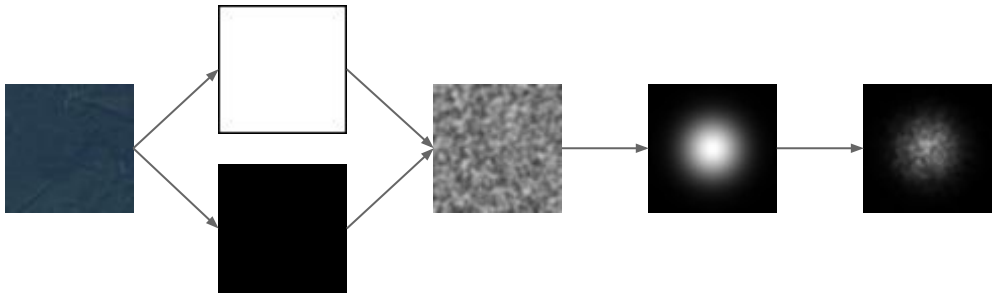
This directory only contains JPG images with Red, Green and Blue bands of images obtained from the Sentinel-2 satellite.

AnnualCrop 3000 files	Forest 3000 files	HerbaceousVegetation 3000 files	Highway 2500 files	Industrial 2500 files
Pasture 2000 files	PermanentCrop 2500 files	Residential 3000 files	River 2500 files	SeaLake 3000 files

Method

Classification: Using a two layer CNN model with a final linear layer to predict the class of the image.

Segmentation: Using a U-Net network to predict the class of each pixel in the image.



Each image has a category. We first create a label image for each category and each input image. The corresponding category will have all 1.0, the other will have all 0.0.

We then apply the model to the input image, which will give us the predicted probabilities for each pixel and each category. We then calculate the loss function. Before backpropagating, we apply a gaussian filter to the loss.

Evaluation

Since the test set only has the categorical label for the whole image, we have to use an averaging algorithm to get the predicted label.

Available algorithms:

- Average all category then select max.
- Select max for each pixel then average.
- Fully connected neural network (freeze segmentation network).

Metrics: accuracy, precision and recall.

Results – Part I

Model: (Classifier) 2 layers convolution network + dense layer.

Accuracy: 68%.

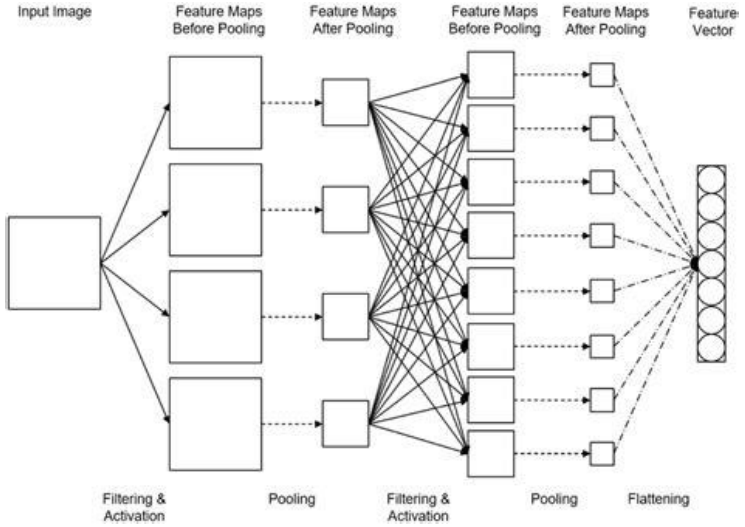


TABLE II: Classification accuracy (%) of different training-test splits on the EuroSAT dataset.

Method	10/90	20/80	30/70	40/60	50/50	60/40	70/30	80/20	90/10
BoVW (SVM, SIFT, k = 10)	54.54	56.13	56.77	57.06	57.22	57.47	57.71	58.55	58.44
BoVW (SVM, SIFT, k = 100)	63.07	64.80	65.50	66.16	66.25	66.34	66.50	67.22	66.18
BoVW (SVM, SIFT, k = 500)	65.62	67.26	68.01	68.52	68.61	68.74	69.07	70.05	69.54
CNN (two layers)	75.88	79.84	81.29	83.04	84.48	85.77	87.24	87.96	88.66
ResNet-50	75.06	88.53	93.75	94.01	94.45	95.26	95.32	96.43	96.37
GoogleNet	77.37	90.97	90.57	91.62	94.96	95.54	95.70	96.02	96.17

Results – Part II

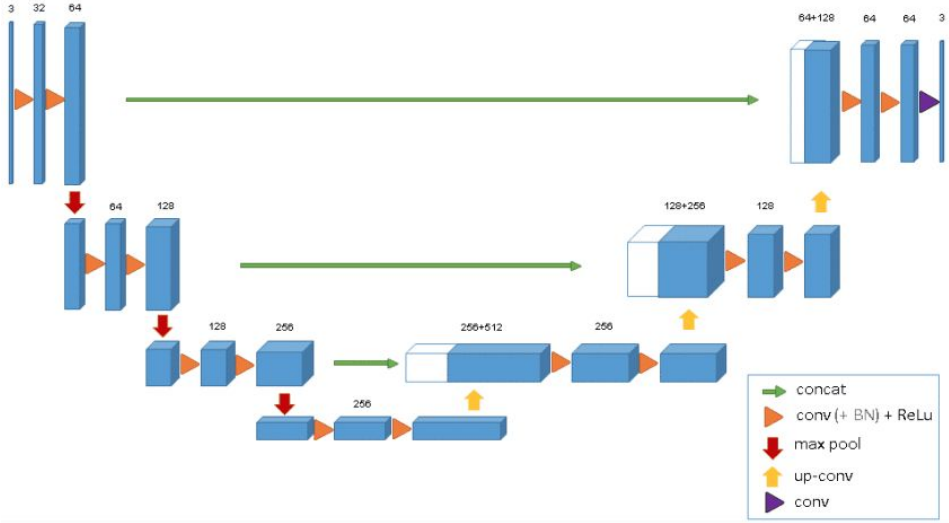
Model: U-Net Architecture.

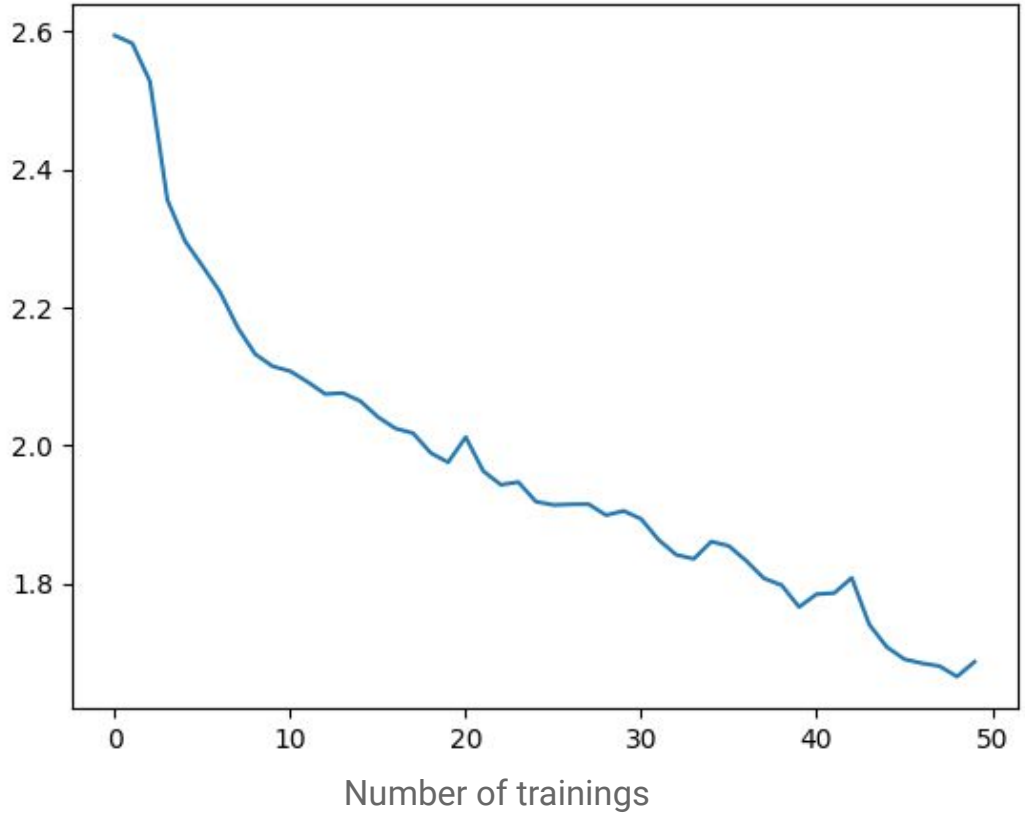
(8 → 12 → 16 → 20 → 20+16 → 16+12 → 12+8 → 8 → 10 classes).

Accuracy: 86%.

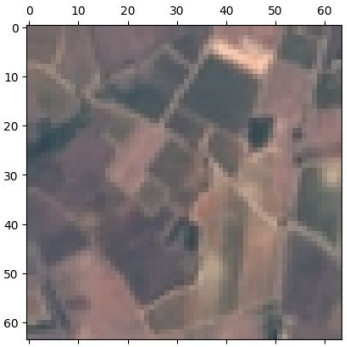
Precision: 93%.

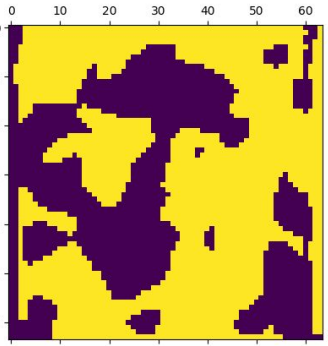
Recall: 91%.



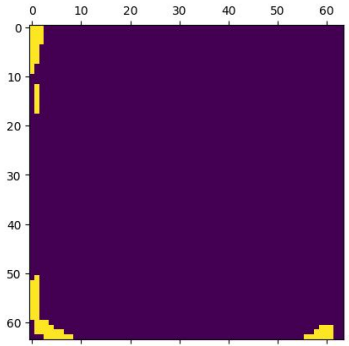


Loss History

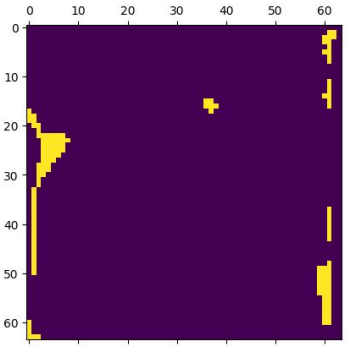




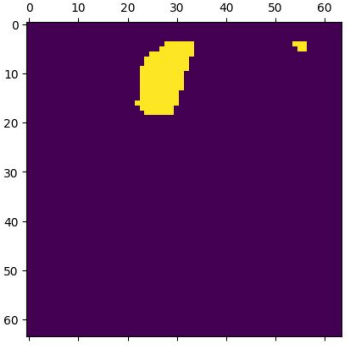
Annual Crop



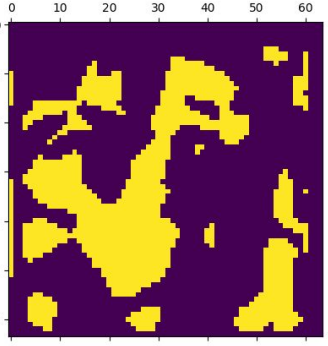
Herbaceous Vegetation



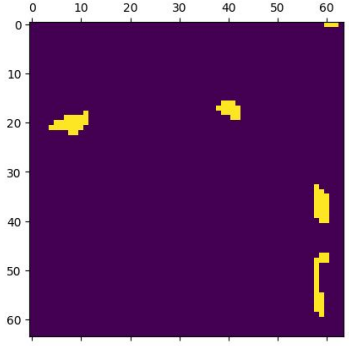
Highway



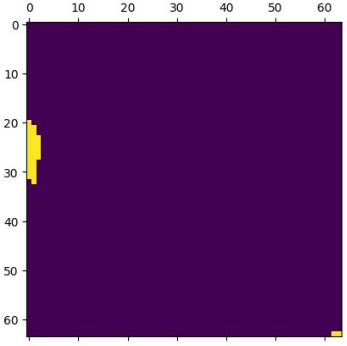
Industrial



Permanent Crop



Residential



River

```python
# %%

import torch
import torch.utils.data
import torch.optim
import torchvision as vision
import torchvision.transforms.v2 as v2
import tqdm
import matplotlib.pyplot as pyplot

import models

def gaussian_filter(kernel_size, sigma=1, muu=0):
    import numpy as np
    # Initializing value of x,y as grid of kernel size
    # in the range of kernel size
    x, y = np.meshgrid(np.linspace(-1, 1, kernel_size),
                       np.linspace(-1, 1, kernel_size))
    dst = np.sqrt(x**2+y**2)
    # Calculating Gaussian filter
    gauss = np.exp(-((dst-muu)**2 / (2.0 * sigma**2)))
    return gauss

Run Cell | Run Above | Debug Cell
# %%

BATCH_SIZE = 2000
TRAIN_SET_SIZE = 0.8

def target_transform(label: int) -> torch.Tensor:
    """Transform label into segmentation matrix."""
    l = torch.zeros((10, 64, 64))
    l[label] = torch.ones((64, 64))
    return l


def target_transform_2(label: int) -> torch.Tensor:
    """Transform label into one-hot encoded."""
    l = torch.zeros(10)
    l[label] = 1
    return l


dataset = vision.datasets.ImageFolder(
    root="dataset",
    transform=vision.transforms.Compose([
        v2.PILToTensor(),
        v2.ToDtype(torch.float32, scale=True),
    ]),
    target_transform=target_transform)

train_set, test_set = torch.utils.data.random_split(dataset, [TRAIN_SET_SIZE, 1 - TRAIN_SET_SIZE])
train_loader = torch.utils.data.DataLoader(dataset=train_set, batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_set, batch_size=BATCH_SIZE, shuffle=False)

print(f"The dataset has {len(dataset)} samples.")
print(f"The train set has {len(train_set)} samples.")
print(f"The test set has {len(test_set)} samples.")
```

```python
unet = models.UNet(n_classes=10).cuda()

# criterion = torch.nn.BCELoss(reduction="mean")
criterion2 = torch.nn.BCELoss(reduction="none")
# optimizer = torch.optim.Adam(cls.parameters(), lr=1e-3)
optimizer = torch.optim.Adam(unet.parameters(), lr=1e-3)

loss_filter = torch.Tensor(gaussian_filter(64)).cuda()

loss_hist = []

Run Cell | Run Above | Debug Cell
# %%

EPOCHS = 50

for epoch in range(EPOCHS):
    total_loss = 0.0
    for inputs, labels in tqdm.tqdm(train_loader):
        inputs = inputs.cuda()
        labels = labels.cuda()
        # y = cls(inputs)
        y = unet(inputs)
        loss = criterion2(y, labels)
        loss = (loss * loss_filter).mean()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # loss_hist.append(loss.item())
        total_loss += loss.item()
    loss_hist.append(total_loss)

pyplot.plot(loss_hist)

tp = 0
tn = 0
fp = 0
fn = 0

for inputs, labels in tqdm.tqdm(test_loader):
    inputs = inputs.cuda()
    labels = labels.cuda()
    # y = cls(inputs)
    y = unet(inputs)
    # acc = torch.argmax(y, dim=1) == torch.argmax(labels, dim=1)
    # accuracy += acc.sum()
    y = torch.argmax(y, dim=1).to(bool)
    labels = torch.argmax(labels, dim=1).to(bool)
    tp += ((y == True) & (labels == True)).sum()
    tn += ((y == False) & (labels == False)).sum()
    fp += ((y == True) & (labels == False)).sum()
    fn += ((y == False) & (labels == True)).sum()

accuracy = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
print(f"Accuracy = {accuracy}")
print(f"Precision = {precision}")
print(f"Recall = {recall}")

Run Cell | Run Above | Debug Cell
# %%

torch.save(
    {
        "model_state_dict": unet.state_dict(),
        "optimizer_state_dict": optimizer.state_dict(),
    },
    "outputs/state.pt")
```

Code