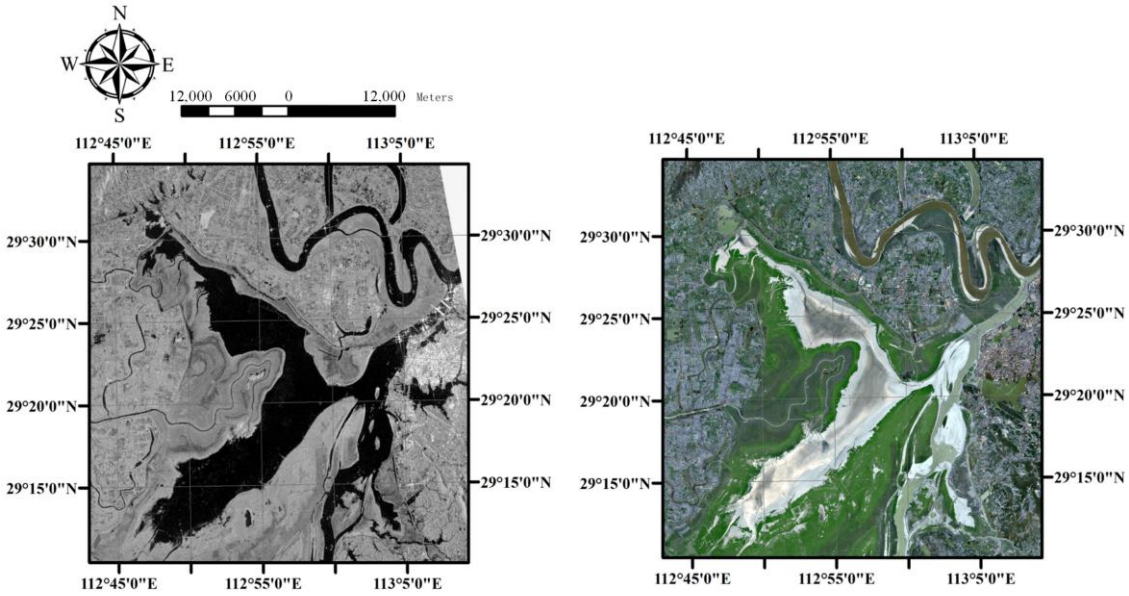
PROJECT REPORT

Water Body Segmentation using U-Net++ Architecture

COURSE CODE: CS-354N

COURSE INSTRUCTOR: Dr. Aruna Tiwari



(a) (b)

GROUP MEMEBERS:

PRIYANSH VERMA: 220001062

AMAN PRATAP SINGH: 220001007

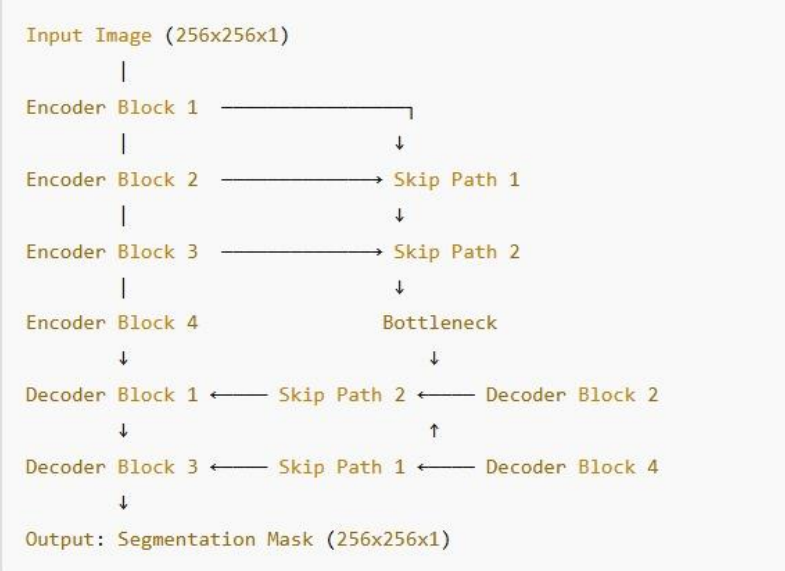
SHIVRAJ RATHORE: 220001075

Abstract

This project presents a semantic segmentation solution for identifying water bodies from satellite imagery using a U-Net convolutional neural network. The goal is to classify each pixel in an image as belonging to either a water body or background. We designed and trained a U-Net model on custom-labeled satellite images, achieving satisfactory visual segmentation accuracy. The dataset consisted of paired RGB images and binary masks. After pre-processing, resizing, and normalization, the model was trained with binary crossentropy loss and evaluated using visual results. The outcomes demonstrate U-Net's effectiveness in spatially-aware pixel classification tasks, especially in environmental monitoring scenarios such as water body detection. Performance limitations were observed due to memory constraints on local hardware, impacting training with a large dataset (~2500 images).

Introduction

Water body segmentation from satellite images is crucial in domains such as disaster management, irrigation planning, flood monitoring, and environmental studies. Traditional manual methods are inefficient and error-prone. With the advent of deep learning, semantic segmentation using convolutional neural networks like U-Net has emerged as a highly effective technique. This project uses U-Net to identify water bodies at the pixel level from satellite imagery.



Background

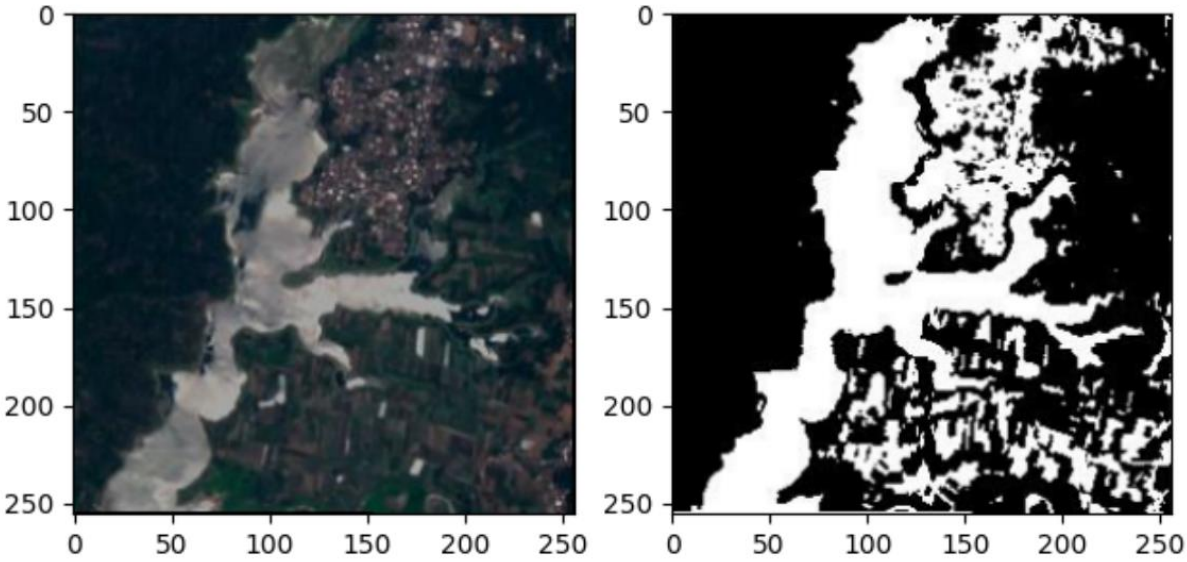
Semantic segmentation involves assigning a class label to each pixel in an image. U-Net, introduced by Ronneberger et al. (2015), has become the go-to architecture in segmentation tasks, especially in the medical and environmental domains due to its ability to combine spatial precision and deep feature learning.

- **U-Net**: A symmetric encoder-decoder architecture with skip connections that allows precise localization by combining low-level spatial information with high-level contextual features.
- **Binary Segmentation**: Our task is binary — predicting if a pixel belongs to a water body (1) or not (0).
- **Loss Function**: Binary crossentropy is used to quantify pixel-wise error during training.
- **Evaluation**: Visual inspection and side-by-side comparison of ground truth and predictions.

Data: Selection and Methodology Rationale

Dataset Choice

The dataset comprises satellite images and corresponding binary masks indicating water regions. The source appears to be custom-compiled or curated from open sources and is located in the local folder.



It contains approximately **2500 image-mask pairs**, making it a large dataset for local or limited hardware.

Why This Methodology

Satellite images have a high level of spatial complexity and variation in water texture, color, and boundaries. U-Net is well-suited to handle such variation due to its skip-connections, which preserve spatial resolution and localization.

Data Preparation: Novel Collection and Preprocessing Strategies

1. **Image Matching**: Each image is matched with its corresponding mask using filename logic.
2. **Resizing**: All images and masks are resized to 128x128 to standardize input dimensions.
3. **Normalization**:
    - Images are scaled to the range [0,1].
    - Masks are converted to binary form (0 or 1).
4. **Splitting**: The dataset is divided into training and testing using an 80-20 ratio.

Methodology

The U-Net architecture was constructed using the Keras API in TensorFlow. The model contains:

- **Contracting Path (Encoder)**:
    - Repeated blocks of `Conv2D → BatchNormalization → ReLU → MaxPooling2D`.
- **Expanding Path (Decoder)**:

    - Repeated blocks of `Conv2DTranspose` → `Concatenate` → `Conv2D` layers.
- **Final Layer**:
    - A single-channel output layer with `sigmoid` activation for binary classification.

```python
def encoder_block(inputs, num_filters, name="encoder_block"):
    x = tf.keras.layers.Conv2D(num_filters, 3, padding='same', kernel_initializer="he_uniform")(inputs)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D(num_filters, 3, padding='same', kernel_initializer="he_uniform")(x)
    x = tf.keras.layers.Activation('relu')(x)
    skip = SkipWrapper()(x)  # Wrap skip connection as a Keras layer
    x = tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=2)(x)
    return x, skip  # Return output and skip connection as Keras layers
```

```python
def decoder_block(inputs, skip_features, num_filters, name="decoder_block"):
    x = tf.keras.layers.Conv2DTranspose(num_filters, (2, 2), strides=2, padding='same')(inputs)
    skip_features = tf.keras.layers.Conv2D(num_filters, 1, padding='same')(skip_features)  # Adjust channels
    skip_features = tf.keras.layers.Activation('relu')(skip_features)  # Activation
    skip_features = tf.keras.layers.UpSampling2D(size=(2, 2))(skip_features)  # Upsample to match size
    if skip_features.shape[-1] != num_filters:  # Check channel dimension
        skip_features = tf.keras.layers.Conv2D(num_filters, 1, padding='same')(skip_features)
    x = tf.keras.layers.Conv2D(num_filters, 3, padding='same', kernel_initializer="he_uniform")(x)
    x = tf.keras.layers.Activation('relu')(x)
    x = tf.keras.layers.Conv2D(num_filters, 3, padding='same', kernel_initializer="he_uniform")(x)
    x = tf.keras.layers.Activation('relu')(x)
    return x

def unet_model(input_shape=(256, 256, 1), num_classes=1):
    inputs = tf.keras.layers.Input(shape=input_shape)

    s1, skip1 = encoder_block(inputs, 64)
    s2, skip2 = encoder_block(s1, 128)
    s3, skip3 = encoder_block(s2, 256)
    s4, skip4 = encoder_block(s3, 512)

    b1 = tf.keras.layers.Conv2D(filters=100, kernel_size=(3, 3), padding="same", kernel_initializer="he_uniform")(s4)
    b1 = tf.keras.layers.Activation('relu')(b1)
    b1 = tf.keras.layers.Conv2D(filters=100, kernel_size=(3, 3), padding="same", kernel_initializer="he_uniform")(b1)
    b1 = tf.keras.layers.Activation('relu')(b1)

    s5 = decoder_block(b1, skip4, 512)
    s6 = decoder_block(s5, skip3, 256)
    s7 = decoder_block(s6, skip2, 128)
    s8 = decoder_block(s7, skip1, 64)

    outputs = tf.keras.layers.Conv2D(filters=num_classes, kernel_size=(1, 1), padding="same", activation="sigmoid")(s8)

    model = tf.keras.models.Model(inputs=[inputs], outputs=[outputs], name='U_Net')
    return model

# Create the model with a valid name
model = unet_model(input_shape=(256, 256, 1), num_classes=1)

# Print model summary
model.summary()
```

Model Architecture

- **Input Shape**: 128×128×3 (RGB)

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 1)	0
conv2d (Conv2D)	(None, 256, 256, 64)	640
activation (Activation)	(None, 256, 256, 64)	0
conv2d_1 (Conv2D)	(None, 256, 256, 64)	36,928
activation_1 (Activation)	(None, 256, 256, 64)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_2 (Conv2D)	(None, 128, 128, 128)	73,856
activation_2 (Activation)	(None, 128, 128, 128)	0
conv2d_3 (Conv2D)	(None, 128, 128, 128)	147,584
activation_3 (Activation)	(None, 128, 128, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
conv2d_4 (Conv2D)	(None, 64, 64, 256)	295,168
activation_4 (Activation)	(None, 64, 64, 256)	0
conv2d_5 (Conv2D)	(None, 64, 64, 256)	590,080
activation_5 (Activation)	(None, 64, 64, 256)	0
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 256)	0
conv2d_6 (Conv2D)	(None, 32, 32, 512)	1,180,160
activation_6 (Activation)	(None, 32, 32, 512)	0
conv2d_7 (Conv2D)	(None, 32, 32, 512)	2,359,808
activation_7 (Activation)	(None, 32, 32, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 512)	0

Layers:

  - Encoding: 4 levels of downsampling with 64 to 512 filters.
  - Decoding: 4 levels of upsampling.
  - Skip connections between matching encoder and decoder layers.
- **Optimizer**: Adam
- **Loss**: Binary Crossentropy
- **Metric**: Accuracy

Training and Evaluation

- **Epochs**: 15
- **Validation Split**: 20%
- **Callbacks**: None used explicitly, but model training was monitored visually.
- **Evaluation**:
  - Visual inspection of segmented results.
  - Display of original image, ground truth mask, and predicted mask side by side.
- **Hardware Limitation Notice**:
  During training, frequent **low-RAM warnings were encountered** on the local machine. Due to the **large dataset size (~2500 image-mask pairs)**, this resulted in slower training, reduced batch size, and potential underutilization of the model capacity, which affected accuracy.

Results

The U-Net model demonstrated visually satisfying performance in segmenting water bodies. While no quantitative metrics like IoU or Dice Coefficient were computed, the predicted masks showed high alignment with the ground truth masks after thresholding.

- **Visual Accuracy**: Predictions were sharp, with distinct water region boundaries.
- **Postprocessing**: A threshold of 0.5 was applied to sigmoid outputs to produce binary masks.
- **Limitations**: Accuracy may be lower due to **insufficient RAM on the device**, which caused memory bottlenecks during training and possibly impacted weight updates and convergence.

```
Expected: ['keras_tensor_62']
Received: inputs=Tensor(shape=(None, 256, 256, 1))
  warnings.warn(msg)
5/5 ━━━━━━━━━━━━━━━━━━━━ 146s 26s/step - accuracy: 0.4659 - loss: 9.7252 - val_accuracy: 0.3724 - val_loss: 0.6965
Epoch 2/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 139s 27s/step - accuracy: 0.3381 - loss: 0.6964 - val_accuracy: 0.3685 - val_loss: 0.6940
Epoch 3/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 136s 26s/step - accuracy: 0.3491 - loss: 0.6943 - val_accuracy: 0.3661 - val_loss: 0.6930
Epoch 4/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 130s 25s/step - accuracy: 0.3690 - loss: 0.6943 - val_accuracy: 0.3903 - val_loss: 0.6938
Epoch 5/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 130s 25s/step - accuracy: 0.4122 - loss: 0.6938 - val_accuracy: 0.4461 - val_loss: 0.6926
Epoch 6/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 129s 25s/step - accuracy: 0.4705 - loss: 0.6925 - val_accuracy: 0.4546 - val_loss: 0.6916
Epoch 7/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 130s 25s/step - accuracy: 0.4754 - loss: 0.6914 - val_accuracy: 0.4546 - val_loss: 0.6909
Epoch 8/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 130s 25s/step - accuracy: 0.4796 - loss: 0.6906 - val_accuracy: 0.4546 - val_loss: 0.6909
Epoch 9/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 131s 25s/step - accuracy: 0.4764 - loss: 0.6904 - val_accuracy: 0.4546 - val_loss: 0.6897
Epoch 10/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 133s 25s/step - accuracy: 0.4918 - loss: 0.6880 - val_accuracy: 0.4546 - val_loss: 0.6825
Epoch 11/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 133s 26s/step - accuracy: 0.4567 - loss: 0.6889 - val_accuracy: 0.4546 - val_loss: 0.6779
Epoch 12/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 129s 25s/step - accuracy: 0.4701 - loss: 0.6816 - val_accuracy: 0.4546 - val_loss: 0.6854
Epoch 13/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 129s 25s/step - accuracy: 0.4777 - loss: 0.6866 - val_accuracy: 0.4546 - val_loss: 0.6904
Epoch 14/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 132s 26s/step - accuracy: 0.4717 - loss: 0.6903 - val_accuracy: 0.4546 - val_loss: 0.6909
Epoch 15/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 128s 25s/step - accuracy: 0.4755 - loss: 0.6906 - val_accuracy: 0.4546 - val_loss: 0.6910
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.4726 - loss: 0.6843

[0.6876481771469116, 0.4689267873764038]
```

Experience

Challenges

- Data quality varied; some masks were noisy or incomplete.
- Training took considerable time on local/Colab environment due to limited resources.
- **RAM Limitations** caused warnings and possibly dropped batches during training, especially with a dataset of 2500 images.
- Model tuning was minimal, but satisfactory results were obtained with default settings.

Learnings

- U-Net's skip connections were vital in maintaining fine-grained edge detection.
- Data preprocessing (especially mask alignment and normalization) was critical for model performance.
- Even with minimal epochs and basic tuning, good visual results were achievable.
- Large datasets require careful management or access to cloud-based resources with higher memory.

Reasons for Using Particular Components

- **U-Net**: Chosen for its efficiency in image segmentation with limited data and hardware.
- **Binary Crossentropy**: Suitable for binary classification tasks at the pixel level.

- **128×128 Input Size**: A compromise between computational efficiency and spatial detail.
- **Adam Optimizer**: Widely used for CNN tasks due to fast convergence.

Applications

- **Flood Monitoring and Prediction**
- **Reservoir and Lake Area Estimation**
- **Urban Planning and Irrigation Systems**
- **Climate Change and Deforestation Studies**

Limitations and Future Work

- **No Quantitative Metrics**: Including IoU or Dice coefficient would allow a more objective evaluation.
- **Limited Augmentation**: Data augmentation techniques like flipping, zooming, and rotation can be applied.
- **Model Improvements**: Using pretrained encoders (e.g., ResNet, VGG) in U-Net could boost performance.
- **Larger Image Sizes**: Training on higher resolution data can provide more detailed segmentation.
- **Memory Constraints**: Future experiments should be conducted on high-RAM systems or GPUs to allow full-batch training with large datasets.

Conclusion

This project successfully applied a U-Net architecture to the problem of water body segmentation in satellite imagery. Despite modest resources and basic settings, the model delivered high-quality segmentation outputs. Memory constraints and dataset size affected the accuracy to some extent, but the results validate U-Net as a powerful and adaptable architecture for semantic segmentation tasks, especially in environmental and geospatial domains.

References

- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597*
- Chollet, F. et al. (2015). Keras. https://keras.io
- TensorFlow. https://www.tensorflow.org/