Aerial Semantic Segmentation Using U-Net Architecture

K S D S SIIDHHU
BTech Electric Engineering 2020
EE20BTECH11020
IIT Hyderabad

Motivation:
Aerial semantic segmentation is a cutting-edge technology that has garnered significant attention in recent years. This advanced computer vision technique involves the partitioning of high-resolution aerial images into meaningful segments, each labeled with the corresponding object or land cover class. The relevance and motivation behind pursuing aerial semantic segmentation projects are manifold and extend across various domains, including environmental monitoring, urban planning, agriculture, disaster management, and beyond. It enables the accurate labeling of objects and land cover in aerial imagery, leading to better decision-making, resource allocation, and improved safety and efficiency in these domains.

I. Introduction

Aerial Semantic Segmentation is basically the semantic segmentation of aerial images. Semantic segmentation is a computer vision technique that classifies image pixels into semantic groups or classes. So in aerial semantic segmentation, the images are divided into type of land, identifying roads, e.t.c.
Following are the steps involved in Aerial Semantic segmentation:

- **Data Aquisition**: I sourced our dataset from Kaggle, comprising 800 images that include both original aerial photographs and their corresponding labeled images.Each image resolution is 6000 x 4000 pixels. Each image in the dataset is paired with its respective ground truth labels, providing a comprehensive set for aerial semantic segmentation. This dataset acquisition from Kaggle ensures a diverse range of scenarios and classes, facilitating robust model training for accurate and detailed segmentation tasks.
- **Data Pre-processing**: I collected the data set from kaggle. Then resized and cropped images for computational efficiency, applied data augmentation for robust learning, and normalized pixel values for standardized input. Class imbalance was addressed, and the dataset was split into training, validation, and testing sets. The final step involved format conversion for compatibility with our chosen deep learning framework. This streamlined preprocessing sets the stage for effective aerial semantic segmentation model training and application.
- **Semantic Labelling(Training)**: Aerial semantic segmentation involves the use of deep learning models, particularly Convolutional Neural Networks (CNNs), to analyze and label each pixel within the image. The model is trained on a labeled dataset where each pixel is associated with a semantic class. During training, the model learns to recognize patterns and features in the imagery that correspond to these classes.
- **Inference(Testing)**: After training, the model will be used to perform inference on new, unlabeled aerial images. It processes the image pixel by pixel and assigns a class label to each pixel based on its learned understanding of the image content.

There are a lot of models to implement semantic segmentation of aerial images. I plan to work on U-Net model

II. EXPLANATION

UNet: U-Net has a symmetric architecture. The encoder network consists of a series of convolutional layers, each of which is followed by a max pooling layer. This downsamples the feature maps at each step, which allows the network to learn more abstract features from the input image. The decoder network consists of a series of transposed convolution layers, each of which is followed by a concatenation layer. This upsamples the feature maps at each step, allowing the network to recover the spatial information lost in the encoder. The decoder network also uses skip connections to fuse the features from different layers of the encoder. This helps improve the segmentation results' accuracy, especially along object boundaries.

- Symmetric architecture with encoder and decoder networks
- Transposed convolution layers for upsampling
- Skip connections to fuse features from different layers

III. U-Net

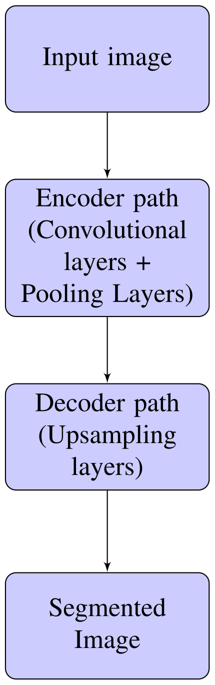
First I started to focus on U-Net model as it's architecture is a bit simple and can be comparatively easy to implement. Before I go into the code, let's first discuss about the architecture of the model.

A. *Architecture*

The architecture of the U-Net model is U-shaped, hence the name.

1) Encoder (Contracting Path):
   - Convolutional layers with increasing spatial resolution and decreasing feature maps. These layers capture hierarchical features from the input image.
   - Max-pooling layers are often used to reduce spatial dimensions while increasing the receptive field.
2) Bottleneck:
   - This is the central part of the U-Net, where the spatial information is reduced to a minimum.
   - It typically consists of a few convolutional layers without pooling.
3) Decoder (Expansive Path):
   - Convolutional layers with decreasing spatial resolution and increasing feature maps. These layers learn to upsample and refine the feature maps.
   - In the decoder, up-convolutions or transposed convolutions are often used to increase the spatial resolution.
   - Skip connections are added from the corresponding layers in the encoder path to provide high-resolution feature maps and help in fine-grained localization.
4) Output Layer:
   - The final layer uses a convolutional layer with a softmax activation function to produce pixel-wise class predictions.

Below is the flowchart of the architecture of the U-Net model


Input image
Encoder path (Convolutional layers + Pooling Layers)
Decoder path (Upsampling layers)
Segmented Image

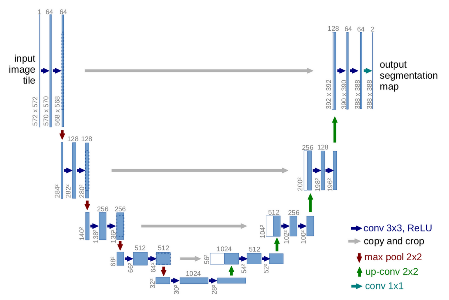


Fig. 1. U-Net architecture

B. *Training Procedure:*

- Load the training dataset.
- Initialize the U-Net model.
- Set the loss function and optimizer.
- For each epoch, iterate over the following dataset.
  - Forward pass: Pass the input image through the U-Net model to obtain a predicted segmentation mask.
  - Calculate the loss: Calculate the loss between the predicted segmentation mask and the ground truth segmentation mask.
  - Backward pass: Backpropagate the loss to update the weights of the U-Net model
- Evaluate the U-Net model on the validation dataset.

C. *Code(Till Now)*

Now that I talked about the architecture and the training procedure of the U-Net model, let's talk about how I coded the U-Net model. I did the coding in Google collaboratory and sometimes in Kaggle (for GPU).

- First, I imported these libraries: tensorflow, keras, numpy, pandas, matplotlib.pyplot, matplotlib.images, os, imageio. I also mounted Google Drive as the training files are rather large, and it isn't practical to repeatedly upload when the runtime has started.
- Then I read the original images, labeled images, and sorted and separated them into training and testing datasets.
- I defined convolutional layers(with parameters- input, number of filters, dropout and max pooling) and upsampling layers(with parameters- 'expansive_input', 'contractive_input', filters, dropout. 'expansive_input', 'contractive_input' are usually outputs from previous convolutional layers.). The kernels used are random gaussian distributions.
- I then defined the U-Net model for shape $96*128*3$ using 6 convolutional layers and 5 upsampling layers previously defined. I included dropout in both convolutional and upsampling layers though I varied the dropout from $0.1-0.4$ and $0.45-0.1$. I also included parameters 'no of filters' and 'no of classes' with default values of 32, 23(23 classes in our dataset), respectively in the model.
- I compiled the model with 'adam' optimizer and sparse categorical entropy.

- I converted the training and testing data into tensors and combined them
- I defined functions process_path (to get training image data and testing image data) and resize to resize training and testing data into images of size $96 * 128 * 3$ using nearest neighbor interpolation. I then got final training and testing datasets in 'final_train_image_data' and 'final_test_image_data'
- I trained the model using the defined 'unet' function for 500 epochs on 32 minibatches. I started out with a loss of 5.5198 and an accuracy of 0.2254; and ended with a loss of 0.2366 and an accuracy of 0.9196
- For each epoch it took a mean(mode) time of $2s$
- I plotted the accuracy with no of epochs. The accuracy fell sharply at around epoch 138 and increased from there again.

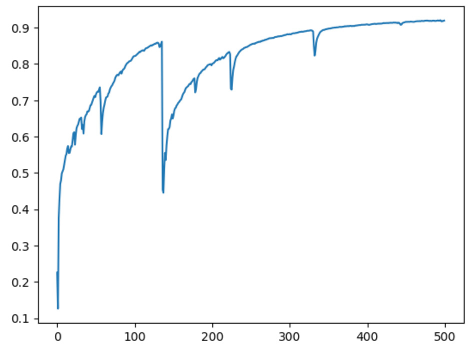


Fig. 2. Accuracy vs Epochs For images of size 96∗128∗3

- I wrote functions to display the input image, its true mask and its predicted mask side by side for n images from the final training dataset.
- I evaluated the model with the final testing dataset. I got a loss of 0.6591398119926453 and an accuracy of 0.8435025811195374
- Next, I decided to run the model for a dataset of images of sizes 800∗1200∗3. But the program crashed as the CPU I had wasn't sufficient to run the model.

D. *Advantages of U-Net for Aerial Semantic Segmentation*

- It is able to learn accurate and detailed segmentation masks, even from small training datasets.
- It is robust to noise and other artifacts in aerial images.
- It can be trained to segment a wide variety of objects in aerial images, such as buildings, roads, and vegetation.

E. *Disadvantages of U-Net for Aerial Semantic Segmentation*

- It can be computationally expensive to train.
- It is sensitive to the hyper-parameters used during training.
- It can be difficult to interpret the predictions of a U-Net model.

IV. References

1) U-Net
2) Aerial Semantic Segmentation Drone Dataset
3) U-Net model