AERIAL SEMANTIC SEGMENTATION

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

POORNA CHANDRA NA
BTech Electric Engineering 2020
EE20BTECH11032
IIT Hyderabad

VIBHUVAN M
BTech Electric Engineering 2020
EE20BTECH11055
IIT Hyderabad

ABHIROOP CHINTALAPUDI
BTech Artificial Intelligence 2020
AI20BTECH11005
IIT Hyderabad

Motivation:
Aerial semantic segmentation is crucial for various applications, such as urban planning, disaster management, agriculture, and autonomous vehicles. 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. We want to run various models trained using different methods and determine which better in which case.

I. Introduction

Aerial Semantic Segmentation is basically the semantic segmentation of aerial images. Semantic segmentation is a computer vision technique that classifies the pixels in images 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**
- **Data Pre-processing**
- **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 is 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.
- **Data Post-processing**

There are a lot of models to implement semantic segmentation of aerial images. We plan to train five pre-trained models using transfer learning

II. Different models for Segmentation

We are currently planning to train five models and determine according to their outputs.
Those models are:

- UNet
- DeepLabV3+ (CNN)
- SegNet
- SegFormer
- Swin

1) **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, which allows the network to recover the spatial information that was lost in the encoder. The decoder network also uses skip connections to fuse the features from different layers of the encoder. This helps to improve the accuracy of the segmentation results, especially along object boundaries.
   - Symmetric architecture with encoder and decoder networks
   - Transposed convolution layers for upsampling
   - Skip connections to fuse features from different layers
2) **DeepLabV3+**: DeepLabV3+ is a semantic segmentation model that builds on DeepLabV3 by adding a simple yet effective decoder module to enhance segmentation results. An encoder that extracts high-level features from the input image, and a decoder that refines the segmentation results by up-sampling the encoder features and fusing them with lower-level features. It also uses atrous convolution in the encoder and decoder modules. Atrous convolution allows the CNN to learn larger receptive fields without increasing the number of parameters. This is important for semantic segmentation because it

allows the CNN to capture context information from the surrounding pixels.

- Modified Xception encoder for efficiency and accuracy.
- Decoder module for refining segmentation results.
- Atrous convolution for learning large receptive fields without increasing parameters.
- Known for its high accuracy and efficiency

3) **SegNet**: The encoder network extracts features from the input image at different levels of abstraction. The decoder network then upsamples the features from the encoder and combines them with the original input image to produce a segmentation map.
   - Fully convolutional network with no fully connected layers.
   - Efficient and easy to train and deploy.
   - Small number of parameters, making it suitable for resource-constrained devices
4) **SegFormer**: SegFormer model that is designed to work on images of any resolution without having an impact on inference performance. It an innovative framework for semantic segmentation that seamlessly combines Transformers with efficient multi-layer perceptron (MLP) decoders.
   It has two main components:
   - Double Paired Modality Encoding (DPE): DPE encodes the input image and its segmentation mask into a joint embedding space. This allows the model to learn long-range dependencies between the image and its segmentation mask.
   - Cross-Modality Attention (CMA): CMA allows the model to attend to different modalities of the input, such as the image and its segmentation mask, to produce more accurate segmentation results.

   Advantages:
   - SegFormer introduces a novel hierarchically structured Transformer encoder that generates multiscale features. Importantly, it eliminates the need for positional encoding, thus avoiding the performance decline associated with interpolating positional codes when testing resolution differs from the training data.
   - SegFormer simplifies the decoder architecture by employing a proposed MLP decoder. This decoder effectively gathers information from various layers, enabling the integration of both local and global attention mechanisms to create robust representations.
5) **Swin**: Swin Transformer uses a hierarchical structure of Transformer blocks, where each block learns features at a different level of abstraction and it is also able to learn long-range dependencies in the input image without having to compute the attention matrix over the entire image. This is important for aerial images, which can be very large and complex.
   - Swin Transformers are able to learn from large and complex aerial images without sacrificing accuracy.
   - Swin Transformers are able to produce high-quality semantic segmentation maps at a relatively fast speed.
   - Swin Transformers are relatively easy to train and deploy.
   - Window attention for efficient attention computation

III. Collecting Data

We downloaded a lot of datasets like iSAID, ISPRS and other kaggle datasets and started to organise them, and finally decided to use the Kaggle Dataset Aerial Semantic Segmentation Drone Dataset.

IV. U-Net

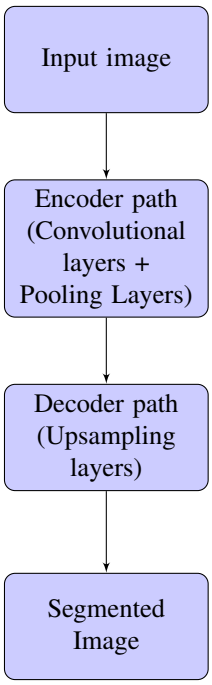
First we started to focus on U-Net model as it's architecture is a bit simple and can be comparatively easy to implement. Before we go into the code, let's first discuss about the architecture of the model.

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



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(Training and Testing)

Now that we talked about the architecture and the training procedure of the U-Net model, lets talk about the how we coded the U-Net model. We did the coding in google colaboratory and sometimes in kaggle(for gpu). We checked the codes on U-Net model online and modified a code from Kaggle

- First, we imported libraries we thought were going to be needed in the code which include tensorflow, keras, numpy, pandas, matplotlib.pyplot, matplotlib.images, os, imageio,cv2. We also mounted google drive as the training files are rather large and it isn't practical to repeatedly upload when the runtime has started.
- We then read the original images and labeled images and sorted and separated them into training and testing dataset.
- We 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.
- We then defined the U-Net model for shape $96 * 128 * 3$ using 6 convolutional layers and 5 upsampling layers previously defined. We included dropout in both convolutional and upsampling layers though we varied the dropout from $0.1 - 0.4$ and $0.45 - 0.1$. We 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.
- We compiled the model with 'adam' optimizer and sparse categorical entropy.
- We converted the training and testing data into tensors and combined them
- We 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. We then got final training and testing datasets in 'final_train_image_data' and 'final_test_image_data'
- We trained the model using the defined 'unet' function for 500 epochs on 32 minibatches. We 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 $2s$
- We plotted the accuracy with no of epochs. The accuracy fell sharply at around epoch 138 and increased from there again.

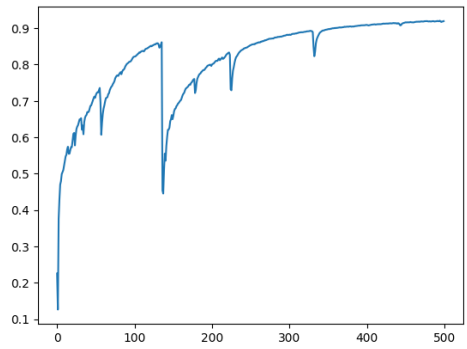


Fig. 1. Accuracy vs Epochs For images of size 96∗128∗3

- We wrote functions to display input image, it;s true mask and it's predicted mask side by side for n images from final training dataset.
- We evaluated the model with final testing dataset. We got a loss of 0.6591398119926453 and an accuracy of 0.8435025811195374
- Next we decided to run the model for dataset of image of sizes $800∗1200∗3$. But the programme crashed as the

CPU we have wasn't sufficient to run the model.

- So we decided to run the model for an image size of $384 * 512 * 3$ for 500 epochs. We also had to reduce our no of batches down to 16 to compensate for our limited CPU. We ended up with a loss of 0.5740513801574707 and an accuracy of 0.864147424697876. For each epoch it took a mean(mode) time of $32s$

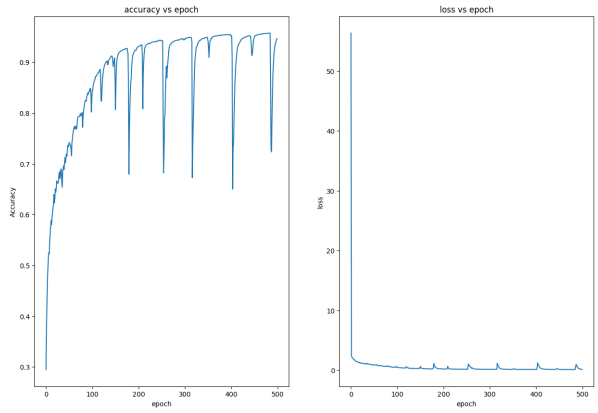


Fig. 2. Accuracy, Loss vs Epochs For images of size 384∗512∗3

- We also ran the model for an image size of $512*512*3$ for 350 epochs. We had further reduce our no of batches to 16 to compensate for our limited CPU. Unfortunately, due to limited time of GPU in colab both me and my friend's time was up and we were unable to finish running the code and stopped at epoch 337. At that epoch we ended up with a loss of 0.1496 and an accuracy of 0.9509.For each epoch it took a mean(mode) time of $42.5s$

D. Advantages of U-Net for Aerial Semantic Segmentation

- It is able to learn accurate and detailed segmentation masks, even from small training data-sets.
- 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.

V. DeepLabV3+

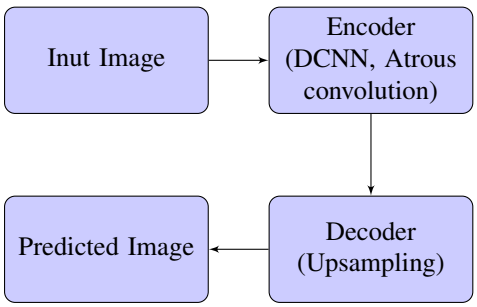
Previously we saw about U-Net model, now lets see about DeepLabV3+ model.
DeepLabV3+ is an extension of the DeepLabV3 architecture. The main improvement of DeepLabV3+ is the use of atrous spatial pyramid pooling (ASPP).

A. Architecture

Below is the architecture of the DeepLabV3+ model.

1) Backbone Network:
   - DeepLabV3+ typically employs a powerful convolutional neural network as its backbone, such as ResNet, MobileNetV2, or Xception.
   - The backbone is responsible for feature extraction from the input image.
2) Atrous Spatial Pyramid Pooling:
   - ASPP is a critical component of DeepLabV3+ that captures multi-scale contextual information.
   - It involves parallel atrous convolutions with different dilation rates and a global average pooling layer.
   - This allows the model to analyze features at different scales, which is essential for semantic segmentation.
3) Encoder-Decoder Structure:
   - DeepLabV3+ uses an encoder-decoder architecture for semantic segmentation.
   - The encoder extracts features from the input image, while the decoder is responsible for upsampling feature maps to generate the final segmentation map.
4) Skip Connections:
   - Skip connections are added from the encoder to the decoder to enhance localization accuracy.
   - These connections allow the model to integrate high-resolution information from earlier layers.
5) Bilinear Interpolation:
   - Bilinear interpolation is used in the decoder to upsample feature maps.
   - It ensures the preservation of fine-grained details during the upsampling process.
6) The Output Layer:
   - The output layer produces the final segmentation map.
   - The number of output channels corresponds to the number of classes you want to segment in your aerial images.

Below is the flowchart of the architecture of the DeepLabV3+ model



B. Training Procedure:

- Load the training dataset.
- Initialize the DeepLabV3+ model.
- Set the loss function and optimizer.
- For each epoch iterate over the following dataset.
  - Forward pass: Pass the input image through the DeepLabV3+ 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 DeepLabV3+ model
- Evaluate the DeepLabV3+ model on the validation dataset.

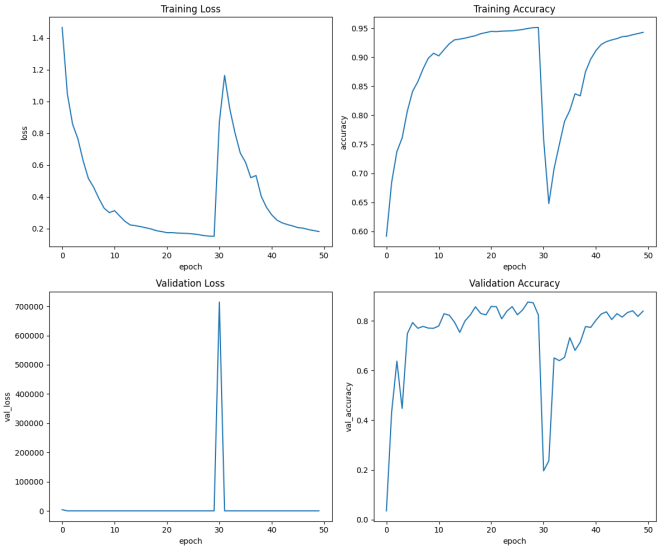


Fig. 3. Accuracy, Loss, validation accuracy and validation loss vs Epochs

C. Code(Training and Testing)

Now that we talked about the architecture and the training procedure of the DeepLabV3+ model, lets talk about the how we coded the model. We did the coding in google colaboratory and sometimes in kaggle(for gpu).
We took reference for the code from here

- We first imported libraries we are going to use in the code which include os, cv2, glob, scipy.io, matplotlib.pyplot, PIL, tensorflow, keras. We also mounted google drive as the training files are rather large and it isn't practical to repeatedly upload when the runtime has started.
- We then read the original images and labeled images and sorted and separated them into training and testing dataset.
- We defined functions read_image, load_data, data_generator to create training, testing and validation datasets consisting of images each of size $512 * 512 * 3$
- We then defined convolution_block(with parameters block_input, num_filters, kernel_size, dilation_rate, padding, use_bias) and DilatedSpatialPyramidPooling(with parameters input usually output from the convolution_block)
- We defined function DeeplabV3Plus(with parameters image_size, num_classes) using the above defined convolution_block, DilatedSpatialPyramidPooling and resnet50 from keras
- We compiled the model using 'adam' optimizer(learning rate=0.001) and 'SparseCategoricalCrossentropy'
- We then proceeded to train the model using the function DeepLabV3Plus for 50 epochs starting with a loss of 1.4664 and accuracy of 0.5917 and ending with a loss of 0.1805 and an accuracy of 0.9427
- We then need to infer from the model(Hasn't been done yet.)

D. Advantages of DeepLabV3+

- It is able to capture multi-scale context information efficiently, which enables it to distinguish between objects with varying scales.
- It is robust to noise and other artifacts in images.
- It is able to achieve state-of-the-art results on a variety of semantic segmentation benchmarks.

E. Disadvantages of DeepLabV3+

- It can be computationally expensive to train.
- It is sensitive to the hyperparameters used during training.
- It can be difficult to interpret the predictions of a DeepLabV3+ model.

VI. SegNet

SegNet is a model that follows an encoder-decoder architecture, where the encoder captures the hierarchical features of the input image, and the decoder generates the segmented output. Before seeing it's code, let's first come to it's architecture.

A. Architecture

1) Encoder:
   - The encoder part of SegNet consists of a series of convolutional and pooling layers. These layers progressively reduce the spatial resolution of the input image while increasing the number of feature channels.
   - SegNet uses a series of convolutional and pooling operations to capture hierarchical features. However, unlike traditional architectures, SegNet saves the indices of the maximum values obtained during max-pooling rather than the actual values.
2) Decoder:

- The decoder part of SegNet is responsible for upsampling the low-resolution feature maps from the encoder. It aims to generate a pixel-wise classification map that corresponds to the original input image.
- SegNet uses the max-pooling indices stored during the encoding phase to perform efficient upsampling. This process helps recreate the spatial details lost during the downsampling.

3) Skip Connections:
   - SegNet incorporates skip connections between corresponding encoder and decoder layers. These connections allow the network to preserve fine-grained details during the upsampling process.
   - Skip connections help address the challenge of information loss during the encoding stage by providing a direct path for information to flow between layers at different resolutions.
4) The final layer of SegNet performs pixel-wise classification. It typically uses softmax activation to assign a probability distribution over the classes for each pixel.
5) The output is a segmented image where each pixel is labeled according to the class it belongs to.

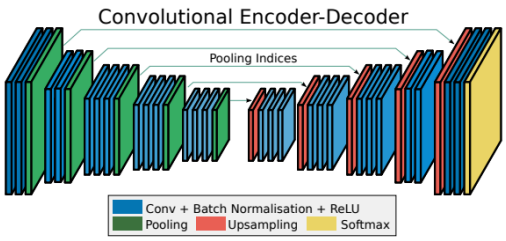


Fig. 4. SegNet

B. Code(Training and Testing)

We did the coding in google colaboratory and sometimes in kaggle(for gpu).
We took reference for the code from here

- First we cloned the repository into our notebook. We basically followed the entire code from the github model.
- We then setup the our python environment according to setup.py in the repository.
- Next we mounted google drive and downloaded our dataset
- We then played around a little with the data
- We then spit the dataset into training set and validation set
- We then created separate folders for all training, validation data.
- From the repository we imported the segnet model and with the image reshape of sizes 384,512 and 512,512
- Next we trained the model for 100 epochs.
- We also set a callback function to stop the training if validation loss is increasing but as our dataset is too small, it was unable to caliculate the validation loss and hence failed to stop the function.
- Next we saved the model.
- Then we predicted the labelled images for some example images and printed them out.

C. Advantages of SegNet

- **Efficiency**: SegNet is a relatively small and efficient network, making it suitable for real-time applications.
- **Precision**: SegNet produces precise segmentation masks that are well-aligned with the input image.
- **Accuracy**: SegNet has achieved state-of-the-art results on a number of benchmark datasets.

D. Disadvantages of SegNet

- Computational burden: Although SegNet is a relatively efficient architecture compared to some other semantic segmentation models, it still has a computational burden that can make it impractical for real-time applications. This is especially true when the input images are large or the number of classes is high.
- Limited feature learning: SegNet's decoder network relies on direct connections to the encoder network to recover spatial information. This can limit the ability of the network to learn new features, especially at higher resolutions.
- Sensitivity to noise: SegNet's pooling and unpooling operations can make it sensitive to noise in the input image. This can lead to inaccurate segmentation masks.

VII. Swin Transformer

A. Architecture

The Swin Transformer architecture introduces a hierarchical design that captures long-range dependencies efficiently. It divides the input image into non-overlapping patches and processes them through a series of hierarchical transformer layers. The key components include patch merging and patch partitioning, enabling the model to capture information at multiple scales.

The hierarchical architecture is characterized by the following components:

- **Patch Embeddings:** The input image is divided into fixed-size patches, and each patch is linearly embedded to generate patch embeddings.
- **Transformer Layers:** Multiple transformer layers are stacked hierarchically. Each layer processes patch embeddings independently, capturing both local and global contextual information.
- **Patch Merging and Partitioning:** To enable the model to capture information at different scales, the architecture incorporates a mechanism for merging and partitioning patches across layers.

- **Class Tokens:** Introduces a class token to aggregate global information, allowing the model to consider the entire image during classification.

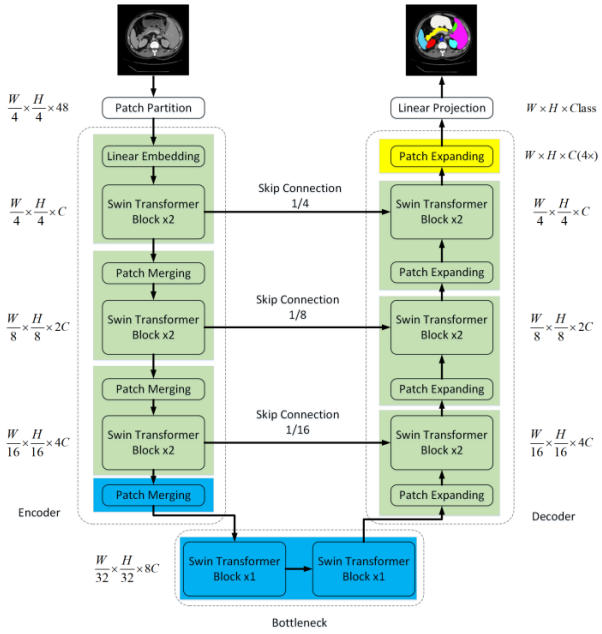


Fig. 5. Enter Caption

B. Code

We did the coding in google colaboratory and sometimes in kaggle(for gpu).
We took reference for the code from here

- Importing Libraries:
  - `os, numpy, pandas, glob, random, cv2, matplotlib.pyplot`
  - `IPython.display, tensorflow, keras, tensorflow.data.Dataset, tensorflow.keras.layers`
  - `tensorflow.keras.applications.mobilenet_v2`
- Clearing Output and Mounting Google Drive.
- Setting Seeds for Reproducibility.
- Defining Image and Mask Directories, Batch Size, Epochs, Learning Rate, Height, and Width.
- Reading Sample Mask, Determining Number of Classes, and Displaying Image Dimensions.
- Loading Image and Mask Paths, Creating a DataFrame, and Splitting into Training and Testing Sets.
- Defining Functions to Read and Augment Images and Masks.
- Creating TensorFlow Datasets for Training and Testing.
- Clearing TensorFlow Backend.
- Creating a Swin UNet Model using the `keras_unet_collection` library.
- Compiling the Model with Adam Optimizer, Focal Loss, and Metrics.
- Training the Model using the Training Dataset and Validating on the Test Dataset.
- Storing Training History in the `history` Variable.
- Making Predictions on a Batch of Test Images.
- Visualizing Original Images, Predicted Masks, and Ground Truth Masks.
- Displaying Sample Results in a Figure.
- we Started to implement the model for 30 epochs but couldn't finish due to constraint in resources

C. Advantages

The Swin Transformer architecture offers several advantages:

- **Efficient Long-Range Dependencies:** The hierarchical design enables the model to capture long-range dependencies efficiently through patch merging and partitioning.
- **Scalability:** The model is scalable to handle input images of varying resolutions without significantly increasing computational complexity.
- **Competitive Performance:** Swin Transformer achieves competitive performance on various computer vision tasks, including image classification and object detection.

D. Disadvantages

Despite its strengths, the Swin Transformer architecture has some limitations:

- **Complexity:** The hierarchical design introduces additional complexity, which may require careful tuning and extensive computational resources.
- **Interpretability:** Interpreting the inner workings of the hierarchical transformer layers may pose challenges, limiting the model's interpretability compared to simpler architectures.

VIII. Results

Fig. 6. Predicted image for UNet

Model	Epochs	Accuracy	Time Taken(Approx)
UNet	150	79.34	120
DeepLabV3+	50	92.97	90
SegNet	100	77.72	110
Swin	10	73.63	120

TABLE I
ACCURACY AND TIME TAKEN FOR EACH MODEL FOR N NUMBER OF EPOCHS

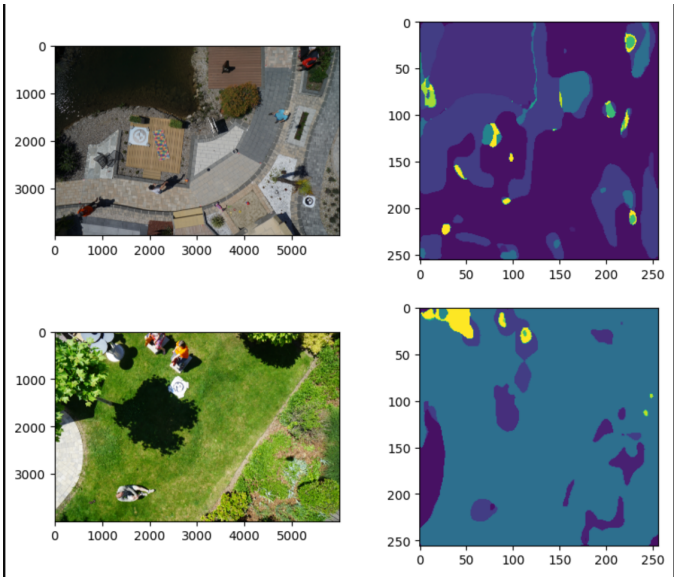


Fig. 7. Enter Caption

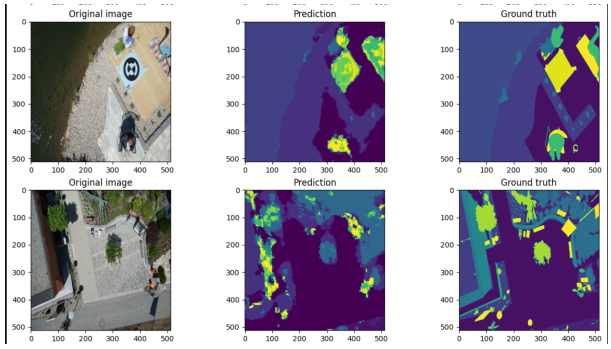


Fig. 8. Predicted image for Swin Transformer

IX. REFERENCES

Unfortunately for DeepLabV3+ we couldn't finish running the code due to the restraint on computational resources.

1) Aerial Semantic Segmentation Drone Dataset
2) ISPRS VAIHINGEN
3) ISPRS TORONTO
4) ISPRS POTSDAM
5) iSAID
6) DEEPLABV3+
7) SegNet
8) U-Net
9) SegFormer
10) Swin Transformer
11) U-Net model SegNet code
12) A Beginner's guide to Deep Learning based Semantic Segmentation using Keras Swin Transformer

X. OTHER FILES SUBMITTED

1) `DeepLabV3+`: The model of DeepLabV3+ we implemented
2) `semantic_0_1`: U-Net model we implemented for image size of $96 * 128 * 3$
3) `semantic_0_2`: U-Net model we implemented for image size of $384 * 512 * 3$
4) `semantic_512_512`: U-Net model we implemented for image size of $512 * 512 * 3$
5) `segnet_384_512`: SegNet model we implemented for image size of $384 * 512 * 3$
6) `segnet_512_512`: SegNet model we implemented for image size of $512 * 512 * 3$
7) `predict_output`: A code to predict images for models we trained for certain images.
8) `swin_unet`: Swin Transformer we implemented for image size of $512 * 512 * 3$