Semantic Segmentation of Aerial Imagery Using Deep Learning Architecture

TECHNICAL DOCUMENTATION

GROUP MEMBERS:

Ajmal Rasheed	**F2022108007**
Hussnain Arshad	**S2022313002**

PROJECT SUPERVISOR:

Dr Mazhar Javed Awan
(Assistant Professor)
Department of Computer Science
University of Management & Technology, Lahore, Pakistan
July 2023

Introduction

Semantic segmentation is a critical task in the field of computer vision, particularly in the analysis of aerial imagery. It involves the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. This task is crucial in various applications, including autonomous driving, precision agriculture, and urban planning. In this study, we introduce a novel approach to semantic segmentation of aerial imagery using an InceptionResNetV2-UNet2 model. The InceptionResNetV2, a convolutional neural network (CNN) that combines the strengths of the Inception networks and the ResNet, serves as the backbone of our model.

Problem Explanation

The primary objective of this project is to develop a robust semantic segmentation framework capable of accurately segmenting aerial imagery into meaningful classes such as roads, buildings, vegetation, water bodies, and other relevant urban elements. The system should leverage state-of-the-art deep learning techniques and advanced image processing algorithms to achieve high accuracy and efficiency.

Dubai Dataset

The MBRSC dataset exists under the CC0 license, available to download. It consists of aerial imagery of Dubai obtained by MBRSC satellites and annotated with pixel-wise semantic segmentation in 6 classes.

There are three main challenges associated with the dataset:

- Class colors are in hex, whilst the mask images are in RGB.
- The total volume of the dataset is 72 images grouped into six larger tiles. Seventy-two images are a relatively small dataset for training a neural network.
- Each tile has images of different heights and widths, and some pictures within the same tiles are variable in size. The neural network model expects inputs with equal spatial dimensions.

Figure-1 depicts a training set *input image* and its corresponding *mask* with superimposed class annotations

Class	Name	Hex #	RGB
0	Unlabelled	9B9B9B	(155, 155, 155)
1	Land	8429F6	(132, 41, 246)
2	Road	6EC1E4	(110, 193, 228)
3	Vegetation	FEDD3A	(254, 221, 58)
4	Water	E2A929	(226, 169, 41)
5	Building	3C1098	(60, 16, 152)

Table - 1 presents each *class name*, corresponding *hex colour* code, and its corresponding color.

Preprocessing

Images must be the same size when fed into the neural network's input layer. Therefore, before model training, images are decomposed into patches.The patch_size chosen is 160 px. There is no ideal patch size; it serves as a hyperparameter that can be experimented with for performance optimization.

Tile	Width (px)	Height (px)	# Patches
1	797	644	8
2	509	544	4
3	682	658	7
4	1099	846	14
5	1126	1058	18
6	859	838	11
7	1817	2061	57
8	2149	1479	49

Table gives tiles; their sizes are the total number of patches created using a size of 256px.

After cropping and patchifying, 1521 images and masks comprise the input dataset

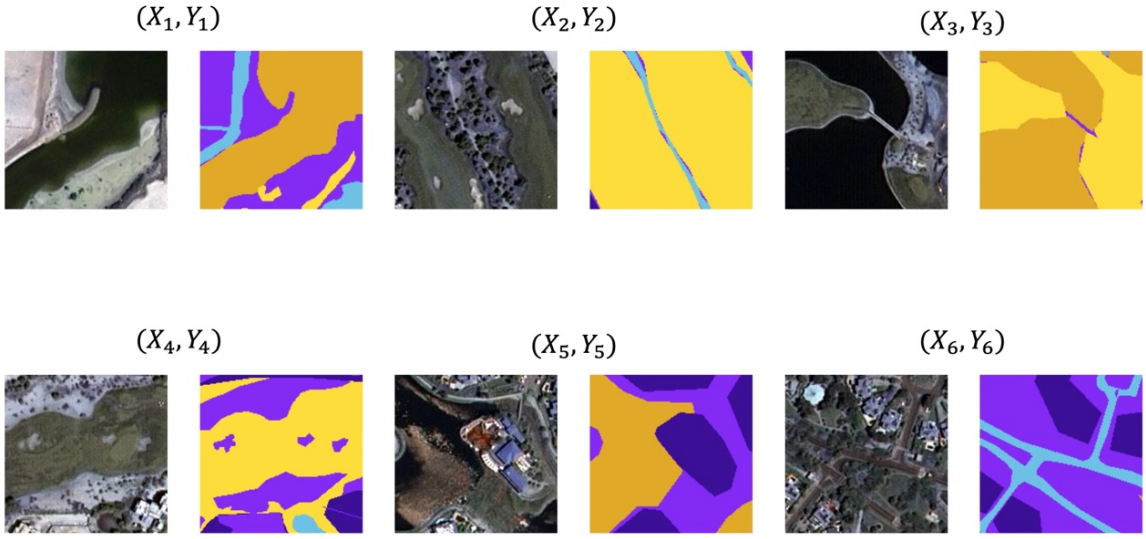


Figure 3 presents six randomly selected image patches with their comparable mask.

There are only 1521 images (having different resolutions) in the dataset, out of which I have used 1187 images (~78%) for training set and remaining 334 images (~22%) for validation set. It is a very small amount of data, in order to artificially increase the amount of data and avoid overfitting, I preferred using data augmentation. By applying these data augmentation techniques, the original dataset of 1521 images can be significantly expanded. In this case, the author states that the dataset was augmented to approximately 9 times its original size, resulting in a training set with 15210 images (1521 + 13689) and a validation set with 334 (original) images. This augmented dataset provides more diverse examples for the model to learn from, reducing the risk of overfitting and improving its generalization ability.

Overall, data augmentation is a valuable technique for dealing with limited training data in semantic segmentation tasks. It helps to create a larger and more diverse dataset, enabling the model to learn robust features and perform better on unseen data.

Data augmentation is done by the following techniques:

- Random Cropping
- Horizontal Flipping
- Vertical Flipping
- Rotation
- Random Brightness & Contrast
- Contrast Limited Adaptive Histogram Equalization (CLAHE)
- Grid Distortion
- Optical Distortion

Here are some sample augmented images and masks from the dataset:

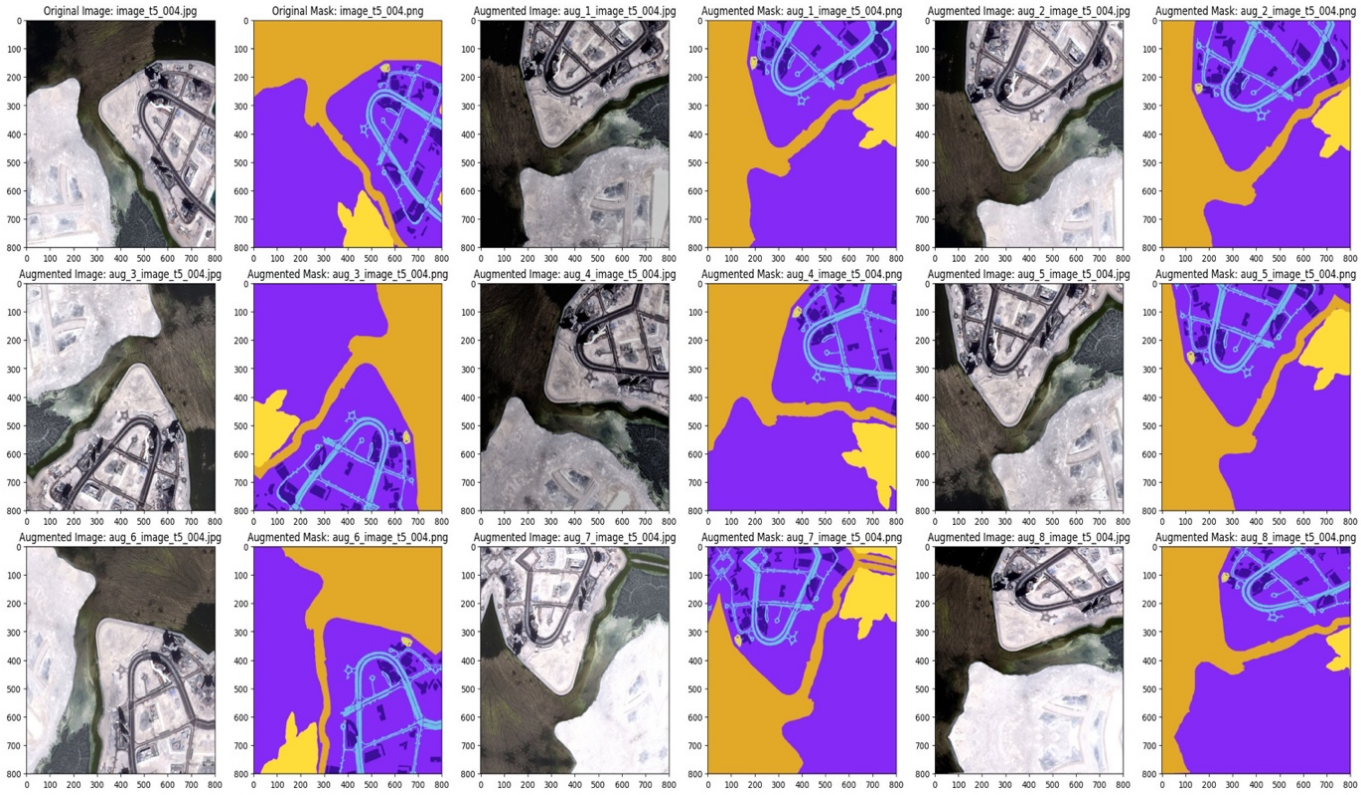
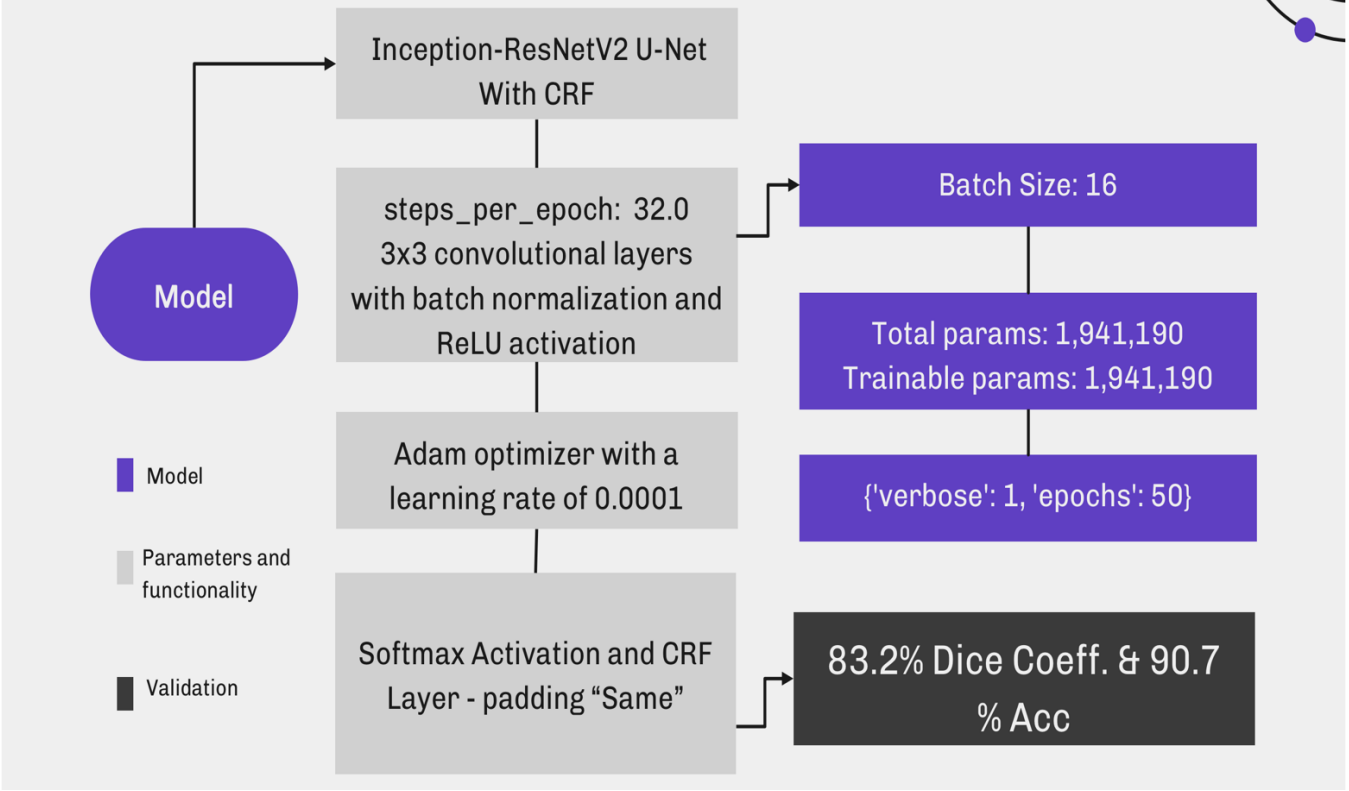


Figure 4 Augmented Images

Model Architecture

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 160, 160, 3) 0
__________________________________________________________________________________________________
rescaling (Rescaling)           (None, 160, 160, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 160, 160, 16) 448         rescaling[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 160, 160, 16) 0           conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 160, 160, 16) 2320        dropout[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 80, 80, 16)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 80, 80, 32)   4640        max_pooling2d[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 80, 80, 32)   0           conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 80, 80, 32)   9248        dropout_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 40, 40, 32)   0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 40, 40, 64)   18496       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 40, 40, 64)   0           conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 40, 40, 64)   36928       dropout_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 20, 20, 64)   0           conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 20, 20, 128)  73856       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 20, 20, 128)  0           conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 20, 20, 128)  147584      dropout_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 10, 10, 128)  0           conv2d_7[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 10, 10, 160)  184480      max_pooling2d_3[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 10, 10, 160)  0           conv2d_8[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 10, 10, 160)  230560      dropout_4[0][0]
__________________________________________________________________________________________________
conv2d_transpose (Conv2DTranspo (None, 20, 20, 128)  82048       conv2d_9[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 20, 20, 256)  0           conv2d_transpose[0][0]
                                                                 conv2d_7[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 20, 20, 128)  295040      concatenate[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 20, 20, 128)  0           conv2d_10[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 20, 20, 128)  147584      dropout_5[0][0]
__________________________________________________________________________________________________
conv2d_transpose_1 (Conv2DTrans (None, 40, 40, 64)   32832       conv2d_11[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 40, 40, 128)  0           conv2d_transpose_1[0][0]
                                                                 conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 40, 40, 64)   73792       concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 40, 40, 64)   0           conv2d_12[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 40, 40, 64)   36928       dropout_6[0][0]
__________________________________________________________________________________________________
conv2d_transpose_2 (Conv2DTrans (None, 80, 80, 32)   8224        conv2d_13[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 80, 80, 64)   0           conv2d_transpose_2[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 80, 80, 32)   18464       concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 80, 80, 32)   0           conv2d_14[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 80, 80, 32)   9248        dropout_7[0][0]
__________________________________________________________________________________________________
conv2d_transpose_3 (Conv2DTrans (None, 160, 160, 16) 2064        conv2d_15[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 160, 160, 32) 0           conv2d_transpose_3[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 160, 160, 16) 4624        concatenate_3[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 160, 160, 16) 0           conv2d_16[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 160, 160, 16) 2320        dropout_8[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 160, 160, 6)  102         conv2d_17[0][0]
==================================================================================================
Total params: 1,421,830
Trainable params: 1,421,830
Non-trainable params: 0
```

Model Configuration



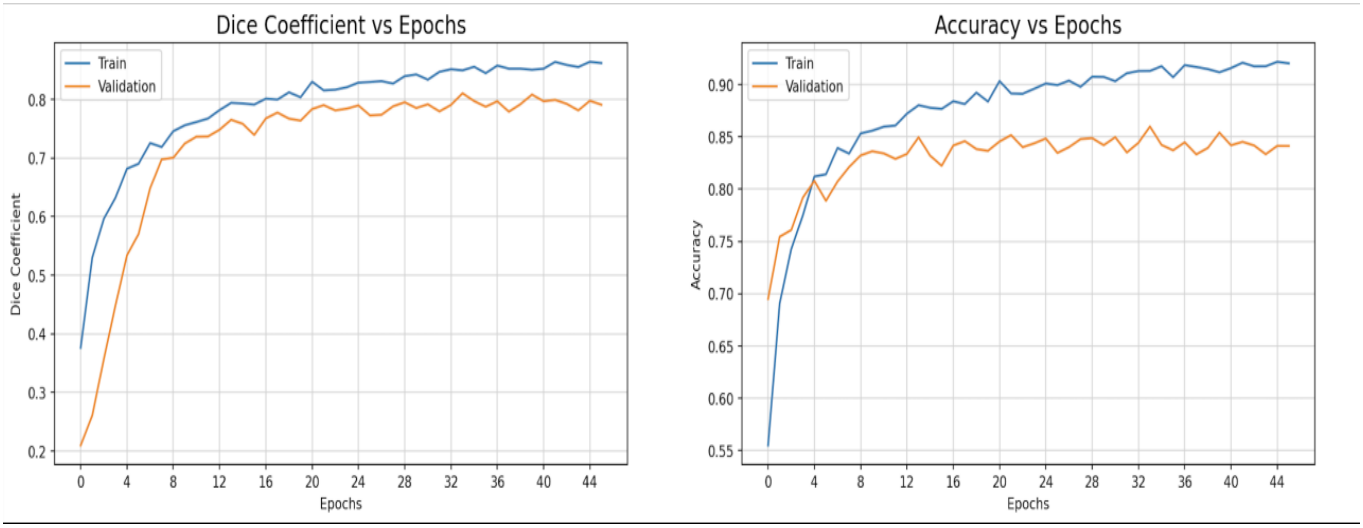
Models Comparison (Dubai Dataset)

Method	Accuracy	Loss Coefficient
U-Net (ResNet Backbone)	0.873	0.743
Multi-UNet	0.842	0.691
Inception-ResNetV2 U-Net	0.861	.432
Inception-ResNetV2 U-Net With CRF	**.907**	**0.832**

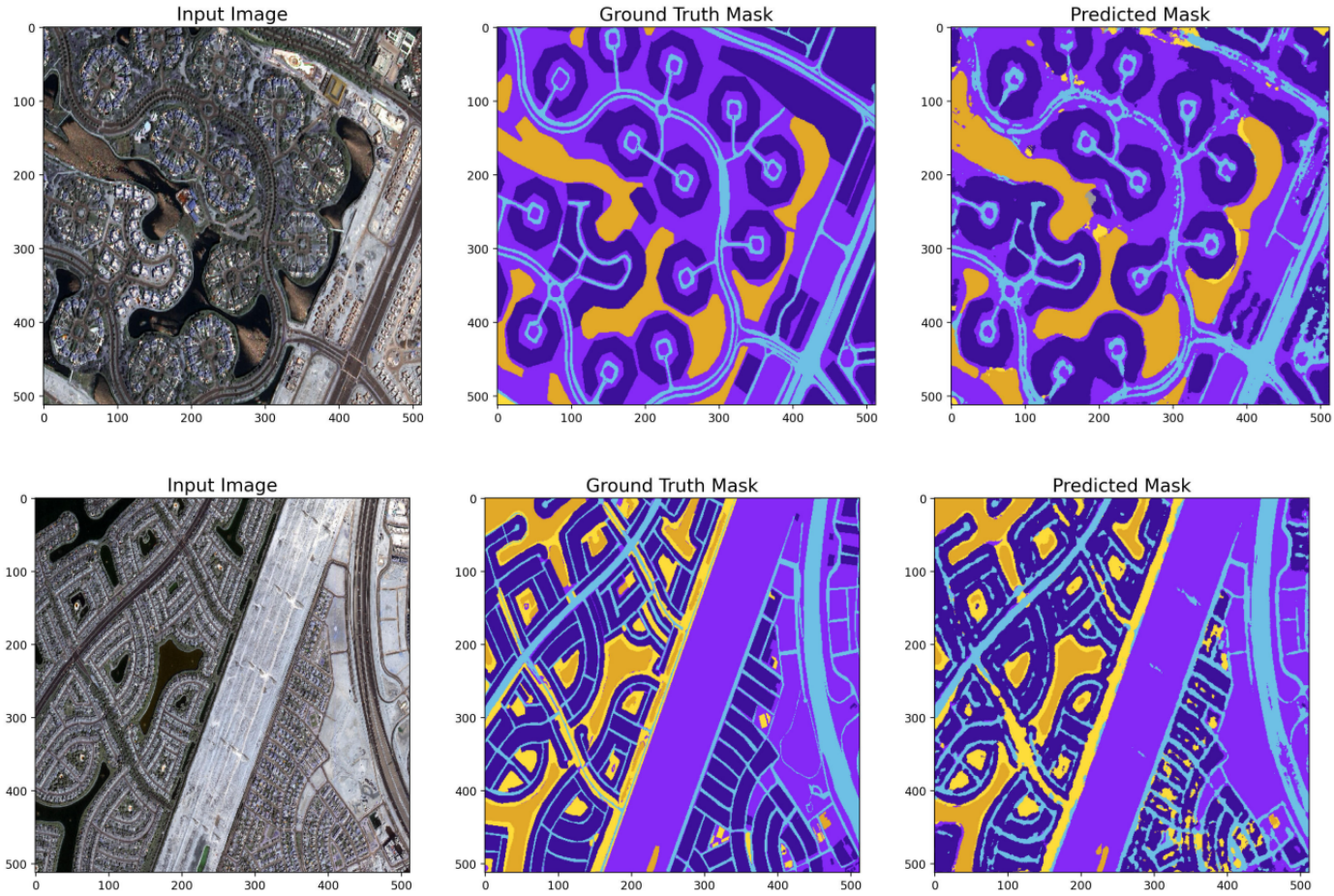
Training Time of Models

Method	Training Time (hrs)	Epochs
U-Net (ResNet Backbone)	4-5	50
Multi-UNet	4.5 - 5	50
Inception-ResNetV2 U-Net	8-9	45
Inception-ResNetV2 U-Net With CRF	**6-8**	**50**

Model Architecture



Predictions



Flask Application

Structure of the Flask App

- **app.py:** This is the main Python file that defines the Flask application. It contains the routes, views, and any other configurations for the app.
- **templates/:** This directory holds the HTML templates used to render the views of the web application.
- **templates/index.html:** An example HTML template that represents the main page or homepage of the web application.

Handling Image Uploads

```
# Get the uploaded image from thse request
image_file = request.files['image']
if not image_file:
    return jsonify({'error': 'No image provided'}), 400

image = Image.open(image_file)
image = image.resize((256,256))
image = np.array(image)
image = np.expand_dims(image, 0)
```

Image Preprocessing

```
# Perform semantic segmentation
model = load_model('segment_model_munet.h5',
                   custom_objects={'dice_loss_plus_1focal_loss': total_loss,
                                   'jaccard_coef': jaccard_coef})
# Replace this with your actual segmentation code
prediction = model.predict(image)
predicted_image = np.argmax(prediction, axis=3)
predicted_image = predicted_image[0,:,:]
segmented_image_new = Image.open('m1.png')

image_np_rescaled = (predicted_image * 255).astype(np.uint8)
segmented_image = Image.fromarray(image_np_rescaled)
# Save the segmented image to a byte stream
output_stream = io.BytesIO()
segmented_image_new.save(output_stream, format='png')
output_stream.seek(0)
```

Model Inference

```
# Your model and '/segment' route go here...

weights = [0.1666, 0.1666, 0.1666, 0.1666, 0.1666, 0.1666]
dice_loss = sm.losses.DiceLoss(class_weights = weights)
focal_loss = sm.losses.CategoricalFocalLoss()
total_loss = dice_loss + (1 * focal_loss)

def jaccard_coef(y_true, y_pred):
  y_true_flatten = K.flatten(y_true)
  y_pred_flatten = K.flatten(y_pred)
  intersection = K.sum(y_true_flatten * y_pred_flatten)
  final_coef_value = (intersection + 1.0) / (K.sum(y_true_flatten) + K.sum(y_pred_flatten) - intersection + 1.0)
  return final_coef_value
```

Displaying Results