Semantic Segmentation of Street Scenes Using Deep Learning

Name	ID
Narden Gerges Adward Amin	2021170576
Nada Abdelmoneem Ahmed	2021170583
Youssef Hany Ezzat Aly	2021170653
Mina Edwar Dawood Elias	2021170565
Dalia Abd Elazim Mohamed	2021170179
Yousef Emad El Din Ibrahim	2021170640
Yousef Essam Ezzat	2021170638

Data preparation process

The data preparation process is crucial for ensuring the semantic segmentation model performs accurately. By defining clear class mappings, preprocessing the data, and applying augmentation, we enhance the robustness and generalization capability of the model.

1. Class Mapping

To ensure pixel-wise classification, we define mappings between RGB values in the mask images and class labels. This helps convert colorful segmentation masks into integer-label masks suitable for model training.

```
class_colors = {
    0: [0, 0, 0],  # Background clutter
    1: [128, 0, 0],  # Building
    2: [128, 64, 128],  # Road
    3: [0, 128, 0],  # Tree
    4: [128, 128, 0],  # Low vegetation
    5: [64, 0, 128],  # Moving car
    6: [192, 0, 192],  # Static car
    7: [64, 64, 0],  # Human
}
```

2. Conversion Function

To transform RGB masks into class label masks, we use the convert_to_label function. This function iterates through the CLASS_MAPPING and assigns the corresponding class label to each pixel based on its RGB value.

```
# Function to map the class index to the corresponding RGB color
def convert_to_label(rgb_mask, class_mapping, color_threshold=30):
    label_mask = np.zeros((rgb_mask.shape[0], rgb_mask.shape[1]), dtype=np.uint8)

    for label, rgb in class_mapping.items():

        distances = np.sqrt(
        np.sum((rgb_mask - np.array(rgb)) ** 2, axis=-1)
        )
        mask = distances < color_threshold
        label_mask[mask] = label

    return label_mask
```

3. Resizing and Normalization

Images and masks are resized to a uniform size and normalized. Masks are converted to class labels using convert_to_label.

```
# Function to resize and normalize images and their corresponding ground truth masks
def preprocess_image(img, mask=None, target_size=(256, 256)):
    img = img.resize(target_size)
    if mask is not None:
        mask = mask.resize(target_size)
        mask = np.array(mask)
        mask = convert_to_label(mask,class_colors)
    img = np.array(img) / 255.0  # Normalize to [0, 1]

    if mask is not None:
        return img, mask
    else :
        return img
```

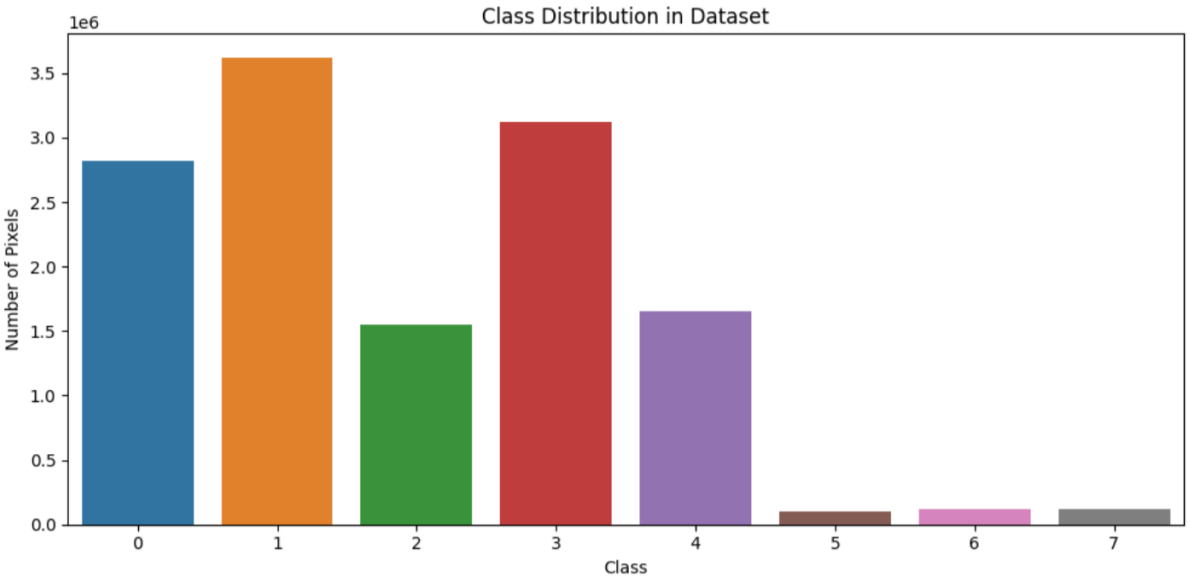
4. Data Augmentation

To increase dataset diversity, we implement data augmentation techniques, such as random flipping. Both images and masks are augmented together to maintain consistency.

```
# Apply augmentations to both image and mask
def augment(img, mask):

    mask_expanded = np.expand_dims(mask, axis=-1)
    image_aug_layer = tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal_and_vertical"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomZoom(0.2)
    ])
    mask_aug_layer = tf.keras.Sequential([
        tf.keras.layers.RandomFlip("horizontal_and_vertical"),
        tf.keras.layers.RandomRotation(0.2),
        tf.keras.layers.RandomZoom(0.2)
    ])
    img_aug = image_aug_layer(tf.expand_dims(img, axis=0))[0]
    mask_aug = mask_aug_layer(tf.expand_dims(mask_expanded, axis=0))[0]
    mask_aug = mask_aug[..., 0]

    return img_aug, mask_aug
```

5. Caught Imbalance Classes

We noticed that there are imbalance between classes by visualize classes and calculate their weights.




```
[0 1 2 3 4 5 6 7]
[2817972 3624430 1551866 3118362 1653306  104589  119222  117453]
{0: 0.581411028924347, 1: 0.4520434937355667, 2: 1.0557612577374593, 3: 0.5254040422503866, 4: 0.9909841251407785,
5: 15.66512730784308, 6: 13.742430088406502, 7: 13.949409551054464}
```

6.Evaluation Functions

We built dice coefficient multi class

```python
def dice_coefficient_multiclass(true, pred, num_classes):
    dice_per_class = []
    for class_id in range(num_classes):
        true_binary = (true == class_id).astype(int)
        pred_binary = (pred == class_id).astype(int)
        intersection = np.sum(true_binary * pred_binary)
        dice = (2 * intersection) / (np.sum(true_binary) + np.sum(pred_binary) + 1e-6)
        dice_per_class.append(dice)
    return np.mean(dice_per_class)
```

7. Model 1(U-NET)

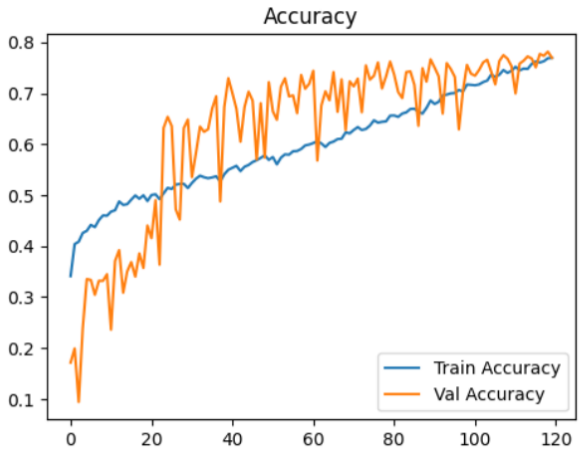
The provided code implements a **U-Net architecture** for image segmentation with **8** output classes. It consists of an **encoder** (feature extraction using convolutional and pooling layers), a **bridge** (bottleneck for feature consolidation), and a **decoder** (up-sampling with skip connections to recover spatial details). The final layer uses a `Conv2D` layer with

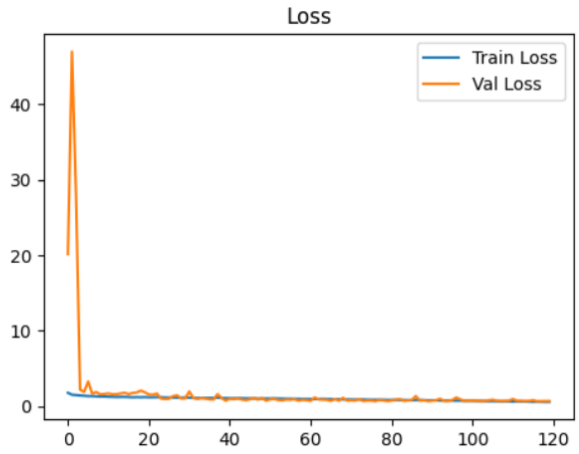
softmax activation to output pixel-wise class probabilities and we use **optimizer** adam **loss** sparse categorical entropy and epochs 120

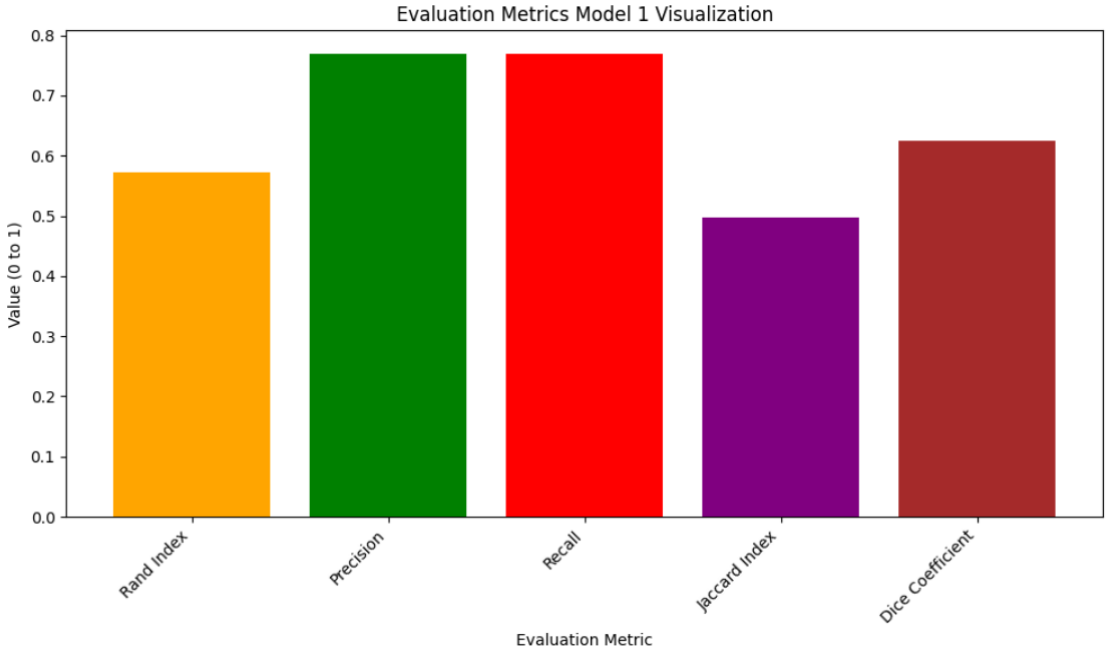
```
num_of_classes =8
# Define the encoder block
def double_conv_block(inputs, num_filters):
  x = Conv2D(num_filters, 3, padding="same")(inputs)
  x = BatchNormalization()(x)
  x = Activation('relu')(x)
  x = Conv2D(num_filters, 3, padding="same")(x)
  x = BatchNormalization()(x)
  x = Activation('relu')(x)
  return x

def encoder_block(inputs, num_filters):
  x = double_conv_block(inputs, num_filters)
  p = MaxPool2D((2,2))(x)
  return x, p

def decoder_block(inputs, skip, num_filters):
  x = Conv2DTranspose(num_filters, (2,2), strides=2, padding="same")(inputs)
  x = Concatenate()([x, skip])
  x = double_conv_block(x, num_filters)
  return x
```

```
def build_unet(input_shape):
    inputs = Input(input_shape)

    # Encoder
    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    # Bridge
    b1 = double_conv_block(p4, 1024)

    # Decoder
    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    # Output layer for 8 classes
    outputs = layers.Conv2D(num_of_classes, 1, padding="same", activation="softmax")(d4)
    model = models.Model(inputs, outputs, name="UNET")
    return model
```

Evaluation of Unet Model :





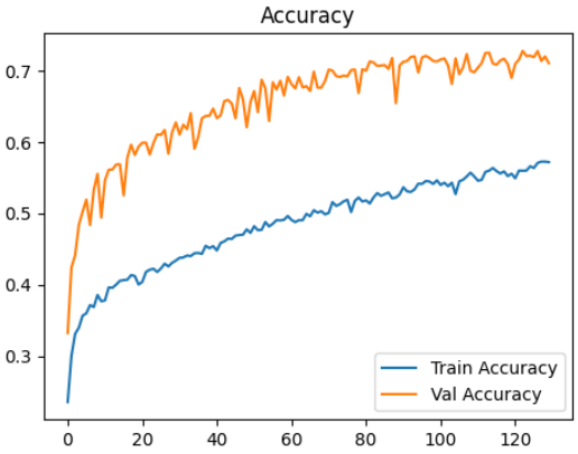


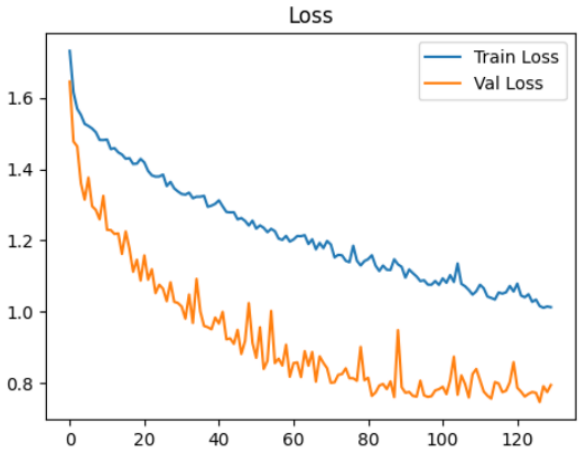
8. Model 2 (FCN)

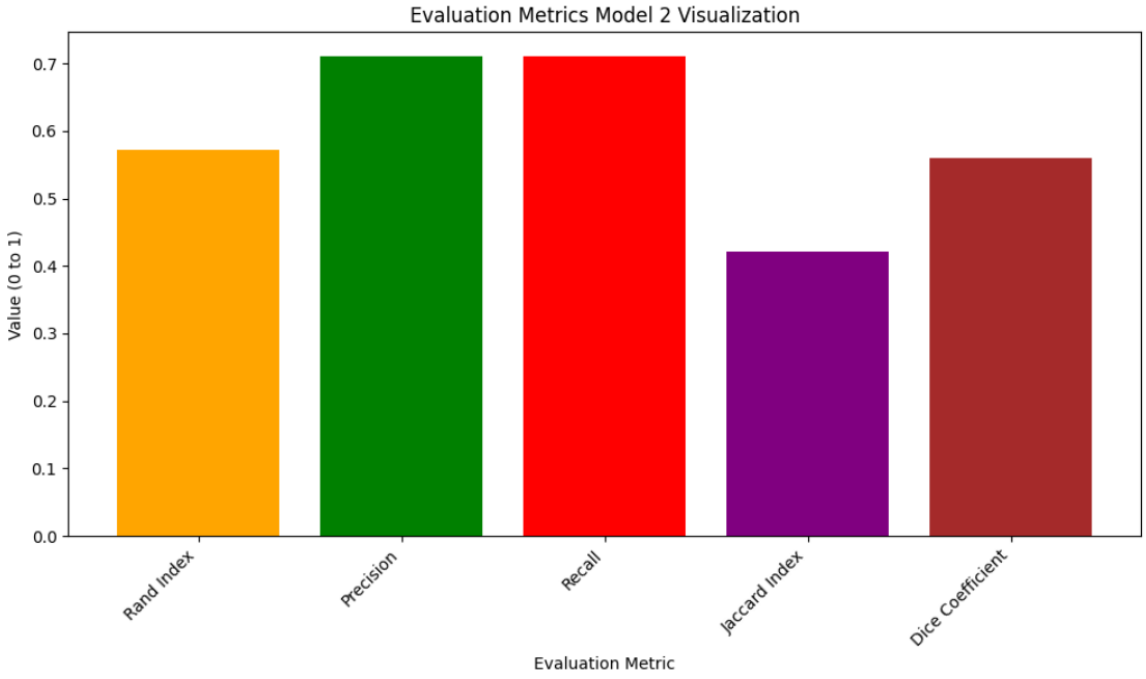
The code implements a Fully Convolutional Network (FCN) for image segmentation with **8** output **classes**. It consists of an encoder with convolutional layers, max-pooling, and dropout for feature extraction, followed by a decoder using transposed convolutions and skip connections to reconstruct the segmentation map. The final output layer uses a Conv2D layer with **softmax** activation for pixel-wise class predictions. The model is trained using the **Adam** optimizer with **sparse categorical cross-entropy** loss for multi-class segmentation over **130** epochs with data augmentation.

```
def fcn(input_shape=(256, 256, 3), num_classes=8):
    inputs = Input(shape=input_shape)
    # Encoder
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    pool1 = MaxPooling2D((2, 2))(conv1)
    Dropout(0.3)
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    pool2 = MaxPooling2D((2, 2))(conv2)
    Dropout(0.3)
    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    Dropout(0.2)
    # Decoder
    up1 = Conv2DTranspose(128, (3, 3), strides=(2, 2), padding='same')(conv3)
    up1 = concatenate([up1, conv2])
    conv4 = Conv2D(128, (3, 3), activation='relu', padding='same')(up1)
    Dropout(0.3)
    up2 = Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same')(conv4)
    up2 = concatenate([up2, conv1])
    conv5 = Conv2D(64, (3, 3), activation='relu', padding='same')(up2)
    Dropout(0.3)
    # Output
    outputs = Conv2D(num_classes, (1, 1), activation='softmax')(conv5)
    return Model(inputs, outputs)
FCN = fcn()
FCN.summary()
```

Evaluation of FCN Model :







9.DeepLab V3

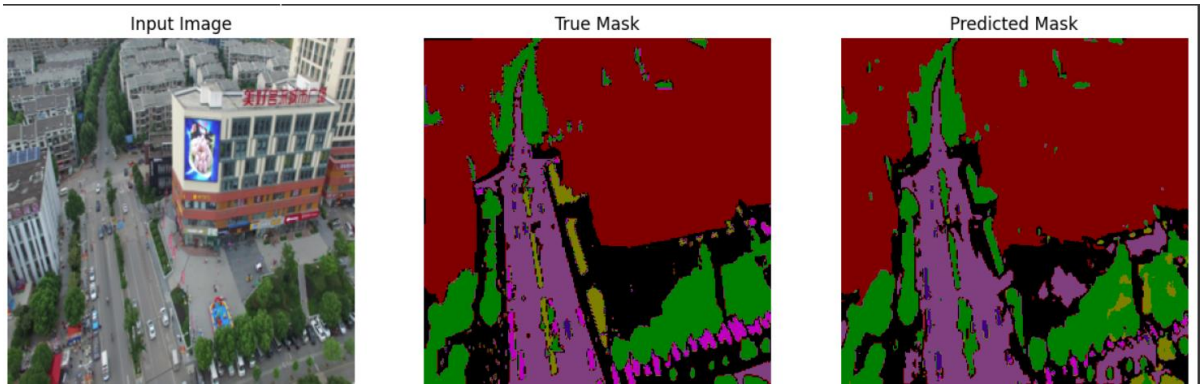
This code combines **U-Net** and **DeepLabV3+** architectures for 8-class image segmentation. The **U-Net** uses an encoder-decoder structure with skip connections, while **DeepLabV3+** leverages **ResNet50** (pretrained on ImageNet) as a feature extractor. After initial training with frozen ResNet50 layers, fine-tuning is performed by unfreezing all layers for 30 epochs. The Adam optimizer and sparse categorical cross-entropy loss are used for optimization.

```
def DeepLabV3Plus(input_shape=(256, 256, 3), num_classes=8):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

    for layer in base_model.layers:
        layer.trainable = False

    unet_model = build_unet(input_shape=input_shape, num_classes=num_classes)

    x = base_model.output

    x = Resizing(256, 256)(x)

    x = Conv2D(3, (1, 1), activation='relu', padding='same')(x)

    unet_output = unet_model(x)

    model = Model(inputs=base_model.input, outputs=unet_output)
    model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

    return model
```

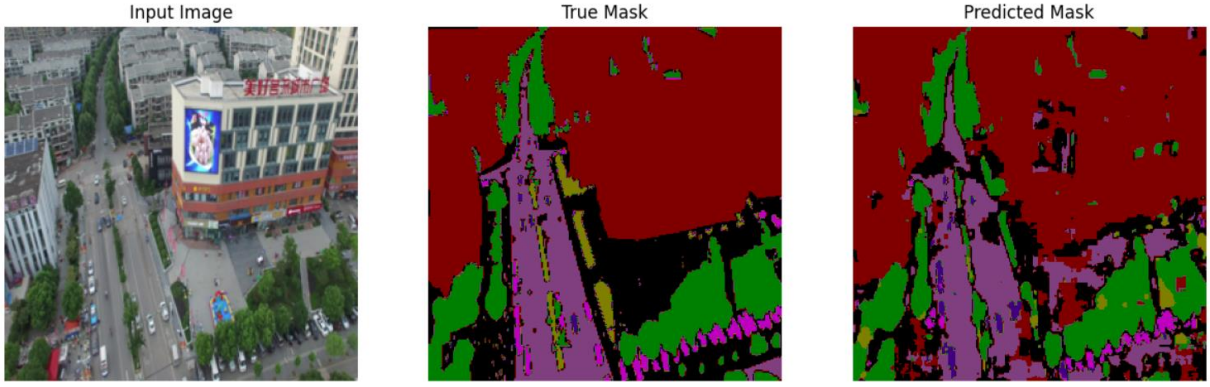
```
##################Bonus5######Fine Tuning
for layer in model3.layers:
    layer.trainable = True
model3.fit(augmented_training_images,augmented_training_masks,epochs=30, batch_size=4
           ,validation_data=(val_images, val_masks))
```

10.Predictions on Validation dataset

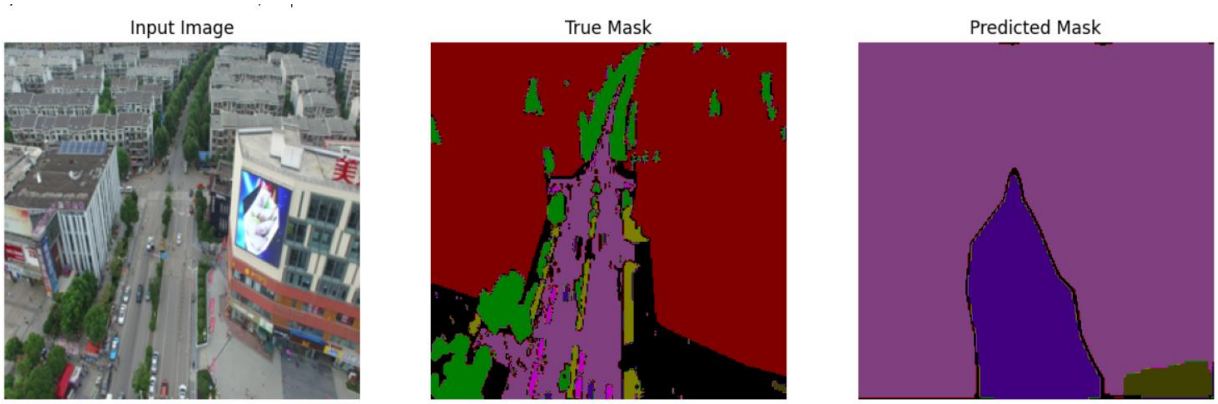
U-NET



FCN



DeepLab V3



11.Show predictions on testing dataset

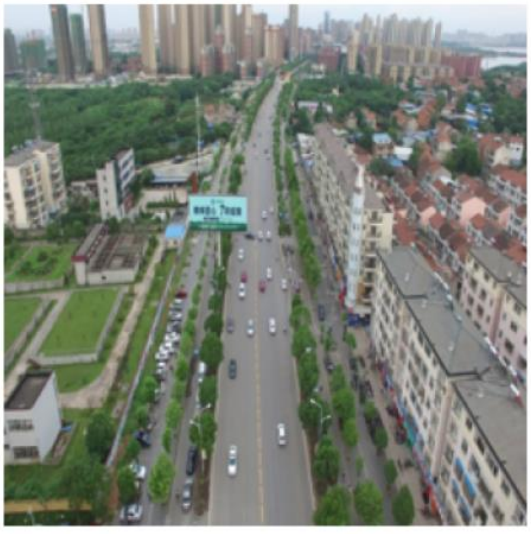
U-NET

Input Image

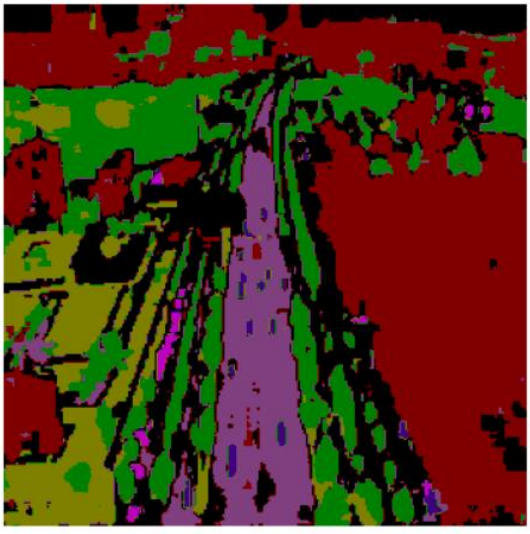
Predicted Mask

FCN

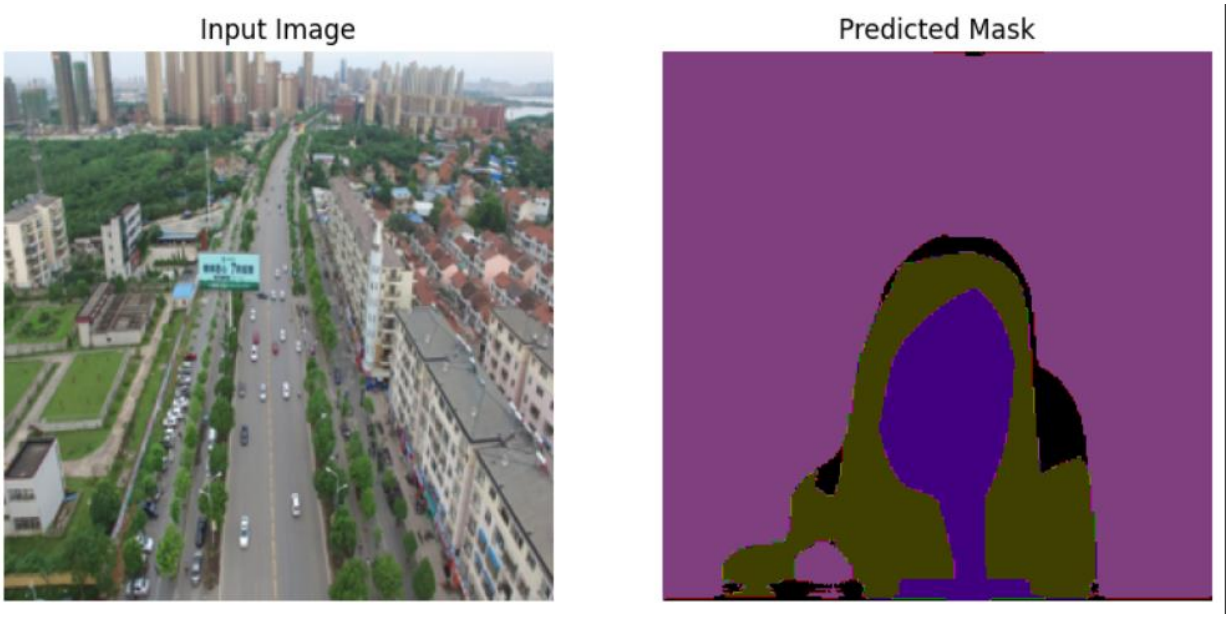
Input Image



Predicted Mask



DeepLab V3



Conclusion

After trying multiple different combinations and adjusting the number of epochs, learning rate, trying different optimizers like SGD and adam, our best model was the U-Net architecture with optimizer{Adam} and a total of 120 epochs, with learning rate{0.001} and with no dropout layers .