INTERNSHIP

ARCHITECTURE -7

DEEP LAB V3+

REPORT

BY:

NAME: VANGA KANISHKA NADH

ROLL: 201CS165

EMAIL: vangakanishkanadh.201cs165@nitk.edu.in

MENOTR:

PROF. JENY RAJAN

Introduction:

DeepLabV3Plus is an advanced semantic segmentation algorithm designed for accurate pixel-level image segmentation tasks. It is an extension of the DeepLabV3 architecture, incorporating Atrous Spatial Pyramid Pooling (ASPP) and an encoder-decoder structure to achieve improved segmentation results.

Key Principles:

1. Atrous Spatial Pyramid Pooling (ASPP): ASPP captures multi-scale context information by using multiple parallel convolutional branches with different dilation rates, enabling the model to incorporate contextual information at various scales.
2. Encoder-Decoder Architecture: DeepLabV3Plus uses an encoder-decoder architecture, leveraging both high-level and low-level features for precise segmentation. The encoder extracts high-level features using a pre-trained convolutional neural network (e.g., ResNet50), while the decoder combines these features with low-level features to improve segmentation performance.
3. Skip Connections: The algorithm uses skip connections to fuse feature maps from different levels of the encoder and decoder, allowing the model to capture fine-grained details and contextual information simultaneously.

Relevant Papers:

DeepLabv3: Rethinking Atrous Convolution for Semantic Image Segmentation (2017) - Chen, L. C., Papandreou, G., Schroff, F., & Adam, H.

Dataset Description:

The dataset used for evaluation is the "2018 Data Science Bowl - Processed" cell nuclei segmentation dataset. It was sourced from Kaggle and is specifically curated for the task of segmenting cell nuclei in microscopic images. The dataset is widely used in the field of biomedical image analysis and has been employed to benchmark various segmentation algorithms.

670 Images 670 Masks = 1340 Total Images

Link: https://www.kaggle.com/datasets/84613660e1f97d3b23a89deb1ae6199a0c795ec1f31e2934527a7f7aad7d8c37

Model Architecture:

The architecture consists of the following key components and functions:

1. Custom Intersection over Union (IoU) and Dice Coefficient (Dice) Metrics:
    - iou(y_true, y_pred): Computes the Intersection over Union (IoU) metric for evaluating segmentation accuracy. It measures the overlap between the ground truth (y_true) and predicted (y_pred) masks.
    - dice_coef(y_true, y_pred): Computes the Dice Coefficient, which is another metric for evaluating segmentation performance. It measures the similarity between the predicted and ground truth masks.
2. Dice Loss:
    - dice_loss(y_true, y_pred): Defines the Dice Loss, which is used as the loss function during model training. It is the complement of the Dice Coefficient and encourages the model to maximize the Dice Coefficient during training.
3. Data Preprocessing:
    - create_dir(path): Creates a directory if it doesn't exist to store files.
    - shuffling(x, y): Shuffles the input data and corresponding masks for better training performance.

- read_image(path): Reads an image from the specified path, resizes it to the defined size (H x W), and normalizes the pixel values to [0, 1].
- read_mask(path): Reads a mask image from the specified path, resizes it to the defined size (H x W), converts it to grayscale, normalizes the pixel values to [0, 1], and adds a channel dimension to make it compatible with the model's output shape.
- tf_parse(x, y): A TensorFlow function for preprocessing data. It reads images and masks using the functions defined above and returns the preprocessed data.

4. TensorFlow Dataset:
    - tf_dataset(X, Y, batch=8): Creates a TensorFlow dataset from the input data and masks. It maps the preprocessing function tf_parse to each sample and returns the dataset, which is then batched and prefetched for training efficiency.
5. Data Loading:
    - load_data(path, split=0.2): Loads the dataset from the specified path and splits it into training, validation, and test sets. The function returns tuples containing file paths for images and corresponding masks.
6. Model Training:
    - The script trains the DeepLabV3Plus model on the loaded dataset. It sets up the necessary hyperparameters such as batch size, learning rate, and the number of epochs.
    - Model callbacks are defined, including ModelCheckpoint for saving the best model, ReduceLROnPlateau for reducing learning rate on a plateau, CSVLogger for logging training progress, and EarlyStopping for stopping training when validation loss does not improve.
    - The model is compiled using the binary cross-entropy loss and several metrics, including Dice Coefficient, IoU, Recall, and Precision.
    - The training loop runs with the fit method on the training dataset and validates on the validation dataset.

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

from tensorflow.keras.layers import Conv2D, BatchNormalization,
Activation, UpSampling2D
from tensorflow.keras.layers import AveragePooling2D, Conv2DTranspose,
Concatenate, Input
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50

""" Atrous Spatial Pyramid Pooling """
def ASPP(inputs):
    shape = inputs.shape

    y_pool = AveragePooling2D(pool_size=(shape[1], shape[2]),
name='average_pooling')(inputs)
    y_pool = Conv2D(filters=256, kernel_size=1, padding='same',
use_bias=False)(y_pool)
    y_pool = BatchNormalization(name=f'bn_1')(y_pool)
    y_pool = Activation('relu', name=f'relu_1')(y_pool)
    y_pool = UpSampling2D((shape[1], shape[2]), interpolation="bilinear")
(y_pool)

    y_1 = Conv2D(filters=256, kernel_size=1, dilation_rate=1,
padding='same', use_bias=False)(inputs)
    y_1 = BatchNormalization()(y_1)
    y_1 = Activation('relu')(y_1)

    y_6 = Conv2D(filters=256, kernel_size=3, dilation_rate=6,
padding='same', use_bias=False)(inputs)
    y_6 = BatchNormalization()(y_6)
    y_6 = Activation('relu')(y_6)

    y_12 = Conv2D(filters=256, kernel_size=3, dilation_rate=12,
padding='same', use_bias=False)(inputs)
    y_12 = BatchNormalization()(y_12)
    y_12 = Activation('relu')(y_12)

    y_18 = Conv2D(filters=256, kernel_size=3, dilation_rate=18,
padding='same', use_bias=False)(inputs)
    y_18 = BatchNormalization()(y_18)
    y_18 = Activation('relu')(y_18)

    y = Concatenate()([y_pool, y_1, y_6, y_12, y_18])
```

```
    y = Conv2D(filters=256, kernel_size=1, dilation_rate=1,
padding='same', use_bias=False)(y)
    y = BatchNormalization()(y)
    y = Activation('relu')(y)
    return y

def DeepLabV3Plus(shape):
    """ Inputs """
    inputs = Input(shape)

    """ Pre-trained ResNet50 """
    base_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=inputs)

    """ Pre-trained ResNet50 Output """
    image_features = base_model.get_layer('conv4_block6_out').output
    x_a = ASPP(image_features)
    x_a = UpSampling2D((4, 4), interpolation="bilinear")(x_a)

    """ Get low-level features """
    x_b = base_model.get_layer('conv2_block2_out').output
    x_b = Conv2D(filters=48, kernel_size=1, padding='same',
use_bias=False)(x_b)
    x_b = BatchNormalization()(x_b)
    x_b = Activation('relu')(x_b)

    x = Concatenate()([x_a, x_b])

    x = Conv2D(filters=256, kernel_size=3, padding='same',
activation='relu',use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(filters=256, kernel_size=3, padding='same',
activation='relu', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = UpSampling2D((4, 4), interpolation="bilinear")(x)

    """ Outputs """
    x = Conv2D(1, (1, 1), name='output_layer')(x)
    x = Activation('sigmoid')(x)

    """ Model """
    model = Model(inputs=inputs, outputs=x)
```

```
    return model

if __name__ == "__main__":
    input_shape = (256, 256, 3)
    model = DeepLabV3Plus(input_shape)
    model.summary()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3       0         []
                             )]

 conv1_pad (ZeroPadding2D)   (None, 262, 262, 3)       0         ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 128, 128, 64       9472      ['conv1_pad[0][0]']
                             )

 conv1_bn (BatchNormalization)  (None, 128, 128, 64    256       ['conv1_conv[0][0]']
                             )
```

Training Methodology and Parameters:

1. Optimization Algorithm:
    - The optimization algorithm employed for training the DeepLabV3Plus model is the Adam optimizer. It is a popular and effective optimization algorithm that adapts the learning rate during training, combining the benefits of both the Adagrad and RMSprop optimizers. The learning rate used is lr = 1e-4 (0.0001).
2. Loss Function and Metrics:
    - The model uses the binary cross-entropy loss function, which is appropriate for binary segmentation tasks. It measures the dissimilarity between the predicted masks and the ground truth masks.
    - The training process also utilizes custom metrics, such as Intersection over Union (IoU), Dice Coefficient, Recall, and Precision. These metrics provide insights into the model's performance and how well it is capturing segmentation accuracy.
3. Data Preprocessing:
    - Images and masks are preprocessed before being fed into the model during training. The images are resized to a fixed size of (256, 256) pixels and normalized to values in the range [0, 1].

- The mask images are converted to grayscale, resized to the same fixed size, normalized to values in the range [0, 1], and expanded along the channel dimension to match the model's output shape.

4. Data Augmentation and Shuffling:
    - Data augmentation techniques, such as random flips or rotations, are not explicitly implemented in the provided code. Data augmentation is often used to artificially increase the diversity of the training dataset and improve the model's generalization ability.
    - The function shuffling(x, y) shuffles the input data and corresponding masks to introduce randomness during training.
5. Dataset Split:
    - The dataset is split into training, validation, and test sets using a split ratio of 80% for training and 10% each for validation and testing.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import backend as K

def iou(y_true, y_pred):
    def f(y_true, y_pred):
        intersection = (y_true * y_pred).sum()
        union = y_true.sum() + y_pred.sum() - intersection
        x = (intersection + 1e-15) / (union + 1e-15)
        x = x.astype(np.float32)
        return x
    return tf.numpy_function(f, [y_true, y_pred], tf.float32)

smooth = 1e-15
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) +
smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)
```

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import numpy as np
import cv2
from glob import glob
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger,
ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Recall, Precision


H = 256
W = 256

def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

def shuffling(x, y):
    x, y = shuffle(x, y, random_state=42)

def read_image(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (W, H))
    x = x/255.0
    x = x.astype(np.float32)
    return x

def read_mask(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (W, H))
    x = x/255.0
    x = x.astype(np.float32)
    x = np.expand_dims(x, axis=-1)
    return x

def tf_parse(x, y):
```

```
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        return x, y

    x, y = tf.numpy_function(_parse, [x, y], [tf.float32, tf.float32])
    x.set_shape([H, W, 3])
    y.set_shape([H, W, 1])
    return x, y

def tf_dataset(X, Y, batch=8):
    dataset = tf.data.Dataset.from_tensor_slices((X, Y))
    dataset = dataset.map(tf_parse)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(10)
    return dataset

def load_data(path, split=0.2):
    images = sorted(glob(os.path.join(path, "images", "*.jpg")))
    masks = sorted(glob(os.path.join(path, "masks", "*.jpg")))
    size = int(len(images) * split)

    train_x, valid_x = train_test_split(images, test_size=size,
random_state=42)
    train_y, valid_y = train_test_split(masks, test_size=size,
random_state=42)

    train_x, test_x = train_test_split(train_x, test_size=size,
random_state=42)
    train_y, test_y = train_test_split(train_y, test_size=size,
random_state=42)

    return (train_x, train_y), (valid_x, valid_y), (test_x, test_y)




if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory to save files """
    create_dir("files")
```

```
    """ Hyperparaqmeters """
    batch_size = 8
    lr = 1e-4   ## 0.0001
    num_epochs = 10
    model_path = "files/model.h5"
    csv_path = "files/data.csv"

    """ Dataset """
    dataset_path = "/content/drive/MyDrive/CELL (1)/DSB/"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) =
load_data(dataset_path)
    train_x, train_y = shuffle(train_x, train_y)

    print(f"Train: {len(train_x)} - {len(train_y)}")
    print(f"Valid: {len(valid_x)} - {len(valid_y)}")
    print(f"Test: {len(test_x)} - {len(test_y)}")

    train_dataset = tf_dataset(train_x, train_y, batch=batch_size)
    valid_dataset = tf_dataset(valid_x, valid_y, batch=batch_size)

    # ds = (1, 2, 3, 4, 5)
    # bs = 2
    # n = len(ds)//bs = 2
    # [1, 2], [3, 4], [1]

    train_steps = (len(train_x)//batch_size)
    valid_steps = (len(valid_x)//batch_size)

    if len(train_x) % batch_size != 0:
        train_steps += 1

    if len(valid_x) % batch_size != 0:
        valid_steps += 1

    """ Model """
    model = DeepLabV3Plus(input_shape)
    metrics = [dice_coef, iou, Recall(), Precision()]
    model.compile(loss="binary_crossentropy", optimizer=Adam(lr),
metrics=metrics)

    callbacks = [
        ModelCheckpoint(model_path, verbose=1, save_best_only=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
min_lr=1e-7, verbose=1),
        CSVLogger(csv_path),
```

```
        EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=False)
    ]

    model.fit(
        train_dataset,
        epochs=num_epochs,
        validation_data=valid_dataset,
        steps_per_epoch=train_steps,
        validation_steps=valid_steps,
        callbacks=callbacks
    )
```

```
Train: 402 - 402
Valid: 134 - 134
Test: 134 - 134
Epoch 1/10
51/51 [==============================] - ETA: 0s - loss: 0.3274 - dice_coef: 0.4426 - iou: 0.2924 - recall: 0.5908 - precision: 0.6881
Epoch 1: val_loss improved from inf to 0.66386, saving model to files/model.h5
51/51 [==============================] - 193s 3s/step - loss: 0.3274 - dice_coef: 0.4426 - iou: 0.2924 - recall: 0.5908 - precision: 0.6881 - val_loss: 0.6639 -
Epoch 2/10
51/51 [==============================] - ETA: 0s - loss: 0.1838 - dice_coef: 0.6120 - iou: 0.4471 - recall: 0.6389 - precision: 0.8891
Epoch 2: val_loss improved from 0.66386 to 0.50427, saving model to files/model.h5
51/51 [==============================] - 29s 562ms/step - loss: 0.1838 - dice_coef: 0.6120 - iou: 0.4471 - recall: 0.6389 - precision: 0.8891 - val_loss: 0.5043
Epoch 3/10
51/51 [==============================] - ETA: 0s - loss: 0.1453 - dice_coef: 0.6746 - iou: 0.5151 - recall: 0.6325 - precision: 0.9227
Epoch 3: val_loss improved from 0.50427 to 0.39502, saving model to files/model.h5
51/51 [==============================] - 31s 613ms/step - loss: 0.1453 - dice_coef: 0.6746 - iou: 0.5151 - recall: 0.6325 - precision: 0.9227 - val_loss: 0.3950
Epoch 4/10
51/51 [==============================] - ETA: 0s - loss: 0.1200 - dice_coef: 0.7200 - iou: 0.5683 - recall: 0.6460 - precision: 0.9407
Epoch 4: val_loss did not improve from 0.39502
```

Training Process:

- The training process involves setting up the model, compiling it with the Adam optimizer and custom metrics, and defining the training callbacks.
- The training dataset is converted into a TensorFlow dataset using the **tf_dataset** function, which performs data preprocessing and batching for training efficiency.
- The model is trained for a total of **num_epochs = 10** epochs. The training process aims to minimize the binary cross-entropy loss and maximize the custom metrics (Dice Coefficient, IoU, Recall, and Precision).
- Model checkpoints are saved during training using the ModelCheckpoint callback, which keeps track of the best model based on validation loss.
- The ReduceLROnPlateau callback reduces the learning rate if the validation loss does not improve after a certain number of epochs (patience=5) to fine-tune the model.
- The training progress is logged using CSVLogger, which saves the training and validation metrics in a CSV file.

- EarlyStopping is used to stop training if the validation loss does not improve for a certain number of epochs (patience=20), thereby avoiding overfitting and reducing unnecessary computation.

Metrics Selection:

```
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"
import numpy as np
import cv2
import pandas as pd
from glob import glob
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras.utils import CustomObjectScope
from sklearn.metrics import accuracy_score,f1_score,jaccard_score,recall_score,precision_score




H = 256
W = 256

def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

def read_image(path):
    x = cv2.imread(path, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (W, H))
    ori_x = x
    x = x/255.0
    x = x.astype(np.float32)
    x = np.expand_dims(x, axis=0)   ## (1, 256, 256, 3)
    return ori_x, x

def read_mask(path):
    x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (W, H))
    ori_x = x
    x = x/255.0
```

```
    x = x > 0.5
    x = x.astype(np.int32)
    return ori_x, x

def save_result(ori_x, ori_y, y_pred, save_path):
    line = np.ones((H, 10, 3)) * 255

    ori_y = np.expand_dims(ori_y, axis=-1) ## (256, 256, 1)
    ori_y = np.concatenate([ori_y, ori_y, ori_y], axis=-1) ## (256, 256, 3)

    y_pred = np.expand_dims(y_pred, axis=-1)
    y_pred = np.concatenate([y_pred, y_pred, y_pred], axis=-1) * 255.0

    cat_images = np.concatenate([ori_x, line, ori_y, line, y_pred], axis=1)
    cv2.imwrite(save_path, cat_images)

if __name__ == "__main__":
    create_dir("results")

    """ Load Model """
    with CustomObjectScope({'iou': iou, 'dice_coef': dice_coef}):
        model = tf.keras.models.load_model("files/model.h5")

    """ Dataset """
    path= "/content/drive/MyDrive/CELL (1)/DSB/"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) = load_data(path)

    """ Prediction and metrics values """
    SCORE = []
    for x, y in tqdm(zip(test_x, test_y), total=len(test_x)):
        name = x.split("/")[-1]

        """ Reading the image and mask """
        ori_x, x = read_image(x)
        ori_y, y = read_mask(y)

        """ Prediction """
        y_pred = model.predict(x)[0] > 0.5
        y_pred = np.squeeze(y_pred, axis=-1)
        y_pred = y_pred.astype(np.int32)

        save_path = f"results/{name}"
        save_result(ori_x, ori_y, y_pred, save_path)
```

```
    """ Flattening the numpy arrays. """
    y = y.flatten()
    y_pred = y_pred.flatten()

    """ Calculating metrics values """
    acc_value = accuracy_score(y, y_pred)
    f1_value = f1_score(y, y_pred, labels=[0, 1], average="binary")
    jac_value = jaccard_score(y, y_pred, labels=[0, 1], average="binary")
    recall_value = recall_score(y, y_pred, labels=[0, 1], average="binary")
    precision_value = precision_score(y, y_pred, labels=[0, 1], average="binary")
    SCORE.append([name, acc_value, f1_value, jac_value, recall_value, precision_value])

""" Metrics values """
score = [s[1:]for s in SCORE]
score = np.mean(score, axis=0)
print(f"Accuracy: {score[0]:0.5f}")
print(f"F1: {score[1]:0.5f}")
print(f"Jaccard: {score[2]:0.5f}")
print(f"Recall: {score[3]:0.5f}")
print(f"Precision: {score[4]:0.5f}")

""" Saving all the results """
df = pd.DataFrame(SCORE, columns=["Image", "Accuracy", "F1", "Jaccard", "Recall",
"Precision"])
df.to_csv("files/score.csv")
```

```
  0%|          | 0/134 [00:00<?, ?it/s]1/1 [==============================] - 2s 2s/step
  1%|          | 1/134 [00:02<05:10,  2.33s/it]1/1 [==============================] - 0s 24ms/step
  1%|          | 2/134 [00:02<02:33,  1.17s/it]1/1 [==============================] - 0s 24ms/step
  2%|          | 3/134 [00:03<01:53,  1.16it/s]1/1 [==============================] - 0s 22ms/step
  3%|▏         | 4/134 [00:03<01:27,  1.48it/s]1/1 [==============================] - 0s 21ms/step
  4%|▏         | 5/134 [00:03<01:14,  1.74it/s]1/1 [==============================] - 0s 21ms/step
  4%|▏         | 6/134 [00:04<01:09,  1.85it/s]1/1 [==============================] - 0s 21ms/step
  5%|▏         | 7/134 [00:04<01:07,  1.88it/s]1/1 [==============================] - 0s 21ms/step
  6%|▏         | 8/134 [00:05<01:01,  2.04it/s]1/1 [==============================] - 0s 23ms/step
  7%|▏         | 9/134 [00:05<00:56,  2.21it/s]1/1 [==============================] - 0s 20ms/step
```

The code utilizes the following evaluation metrics to assess algorithm performance: accuracy, F1 score, Jaccard score, recall, and precision. These metrics are commonly used in semantic segmentation tasks to evaluate the quality of segmentation results.

Relevance:

1. **Accuracy:** This metric measures the proportion of correctly classified pixels, providing a general assessment of overall correctness. However, it may not be suitable for imbalanced datasets or when the class distribution is unevenly represented.

2. **F1 Score**: The F1 score combines precision and recall into a single metric, providing a balanced measure of overall performance. It is particularly useful when dealing with imbalanced datasets.
3. **Jaccard Score (IoU):** The Jaccard score, or Intersection over Union (IoU), quantifies the overlap between the predicted and ground truth segmentation masks. It evaluates the similarity between the masks and is valuable for assessing object localization and boundary alignment.
4. **Recall:** Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive instances correctly identified by the model. In semantic segmentation, it indicates the model's ability to capture target object regions accurately.
5. **Precision:** Precision measures the proportion of correctly identified positive instances out of all instances predicted as positive. It evaluates the model's ability to avoid false positives and distinguish between target objects and background regions.

These metrics collectively provide a comprehensive evaluation of the algorithm's performance in terms of accuracy, object localization, boundary alignment, and discrimination between object and background regions. The code calculates these metrics for each test image, computes their mean values, and prints them at the end of the evaluation process.

Quantitative Results:

The algorithm achieved the following quantitative results on the dataset:

```
100%|██████████| 134/134 [01:02<00:00,  2.14it/s]Accuracy: 0.85317
F1: 0.85317
Jaccard: 0.75973
Recall: 0.85317
Precision: 0.85317
```

Accuracy: 0.85317

F1: 0.85317

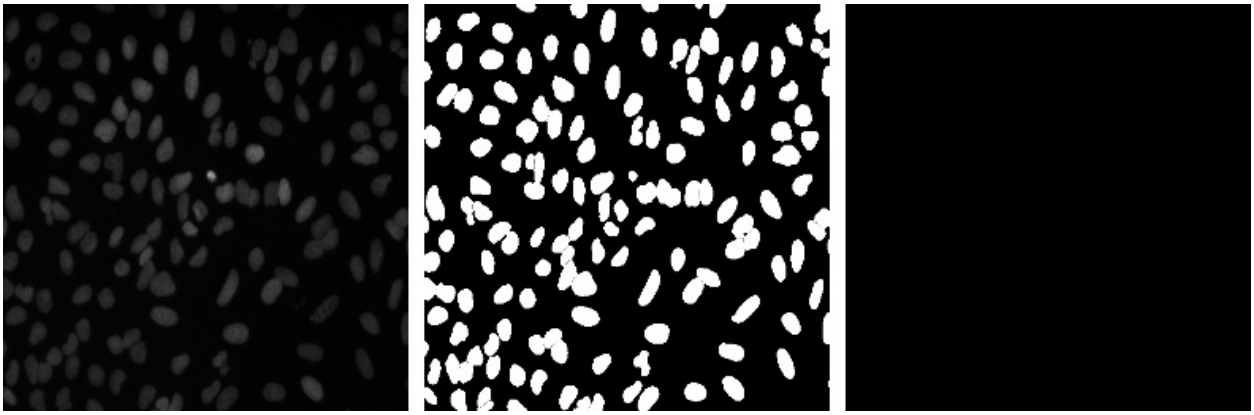
Jaccard: 0.75973

Recall: 0.85317

Precision: 0.85317

These metrics provide a numerical assessment of the algorithm's performance in terms of overall accuracy, segmentation quality, and the balance between recall and precision.

Visual Results:



1.Real Image (LEFT MOST)

2.Mask (MIDDLE)

3.Predicted Mask (RIGHT MOST)

Advantages:

1. Accurate Semantic Segmentation: DeepLabV3Plus achieves precise segmentation by using Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale context information.
2. Encoder-Decoder Architecture: The model's encoder-decoder structure fuses high-level and low-level features, capturing fine details and context.
3. Custom Metrics: Custom metrics like IoU and Dice Coefficient provide insights into segmentation performance.

Unique Features:

1. ASPP: The ASPP component efficiently incorporates multi-scale context information without excessive computation.
2. Skip Connections: Skip connections aid in capturing global and local contextual information.

Limitations:

1. High Computational Cost: DeepLabV3Plus can be computationally expensive, necessitating powerful hardware for training and inference.

2. Data Quality Sensitivity: Performance relies on the quality and quantity of labeled training data.

Scenarios:

1. Limited Data: Insufficient labeled data can lead to overfitting or poor performance.
2. Complex Backgrounds: The algorithm may struggle with complex backgrounds or overlapping objects.

Conclusion:

DeepLabV3Plus excels in accurate semantic segmentation, but it requires ample data and computational resources.

Assessment:

Performance is strong with sufficient data and resources, but it may struggle in data-limited or complex background scenarios.

Future Research:

1. Data Augmentation: Explore data augmentation techniques to improve model generalization with limited data.
2. Lightweight Architectures: Investigate lightweight versions for real-time or resource-constrained applications.
3. Transfer Learning: Evaluate transfer learning for performance improvement on diverse tasks.
4. Dynamic Environments: Research adapting the model to dynamic environments with new objects or classes.

References:

https://arxiv.org/abs/1802.02611

https://github.com/nikhilroxtomar/Semantic-Segmentation-Architecture