در این پروژه، از یک پایگاه داده شامل تصاویر برگ ها و ماسک های متناظر آن ها استفاده می‌شود. هدف از این پروژه، سگمنت بندی تصاویر برگ است، یعنی تفکیک برگ از سایر بخش های تصویر مانند پس زمینه یا اطراف. این فرایند، به ما کمک می‌کند تا تمرکز خود را بر روی برگ قرار داده و آن را از قسمت های دیگر تصویر جدا کنیم.
برای سگمنت بندی تصاویر برگ از پس زمینه، روش های مختلفی وجود دارند. این روش ها شامل الگوریتم‌های پردازش تصویر مبتنی بر رنگ و مورفولوژی، لبه یابی، روش های مبتنی بر بینایی ماشین و یا شبکه های عصبی عمیق می‌شوند. با استفاده از این روش ها، قسمت های مربوط به پس زمینه از برگ تفکیک شده و برگ با دقتی از تصویر استخراج می‌شود.

در این پروژه، از شبکه های عصبی عمیق برای سگمنت بندی تصاویر استفاده شده است. در ادامه هر یک از سلول های کد در ژوپیتر را به صورت مجزا با خروجی بررسی می‌کنیم.

بررسی کد:

در سلول زیر کتابخانه های مورد نیاز را import کردیم. در ادامه به طور خلاصه کاربرد هایی که از آن ها در این پروژه استفاده شد را بررسی می‌کنیم.

از os برای ارتباط با سیستم عامل برای دسترسی به فایل ها و پوشه ها، از zipfile برای اکسترکت فایل ها از یک فایل زیپ، از shutil برای کپی، حذف و جا به جایی فایل و پوشه، از PIL و Image برای بارگذاری و ذخیره تصویر، از matplotlib برای ترسیم نمودار، از numpy برای تولید آرایه های چند بعدی و عملیات جبری، از cv2 برای استفاده از ابزار های OpenCV خواندن و نمایش تصویر و اعمال فیلتر های تصویری استفاده کردیم.

tensorflow یک کتابخانه محاسبات عددی را برای آموزش و استفاده از شبکه های عصبی فراهم می‌کند.

ImageDataGenerator داده های تصویری تغییر یافته را برای آموزش شبکه عصبی فراهم می‌کند مانند چرخش، زوم و نمونه برداری تصادفی.

Model از tensorflow مدل شبکه عصبی را تعریف و ساختار آن را تعیین می‌کند.

Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate از ماژول های مختلف لایه های شبکه عصبی در tensorflow استفاده می‌کند مانند ورودی، لایه های کانولوشن، لایه های pooling و لایه های اتصال و upsample

از ماژول Adam در tensorflow استفاده می‌شود که یک بهینه ساز برای آموزش مدل شبکه عصبی را فراهم می‌کند.

از ماژول Callback در tensorflow استفاده می‌شود که کالبک های مختلفی را برای استفاده در آموزش شبکه عصبی فراهم می‌کند، مانند کالبک های ذخیره سازی مدل و پیگیری عملکرد مدل.

```
import os
import zipfile
import shutil
from PIL import Image
import random
import matplotlib.pyplot as plt
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import Callback
```

در سلول زیر با استفاده از دستور`wget` ، فایل پایگاه داده ما از این آدرس از اینترنت دریافت می‌شود و در مسیر مشخص شده ذخیره می‌شود.

```
!wget -P /content/leaf-segmentation/raw-dataset http://liris.univ-lyon2.fr/reves/documents/Extrait_Pl@ntLeaveI_segmente_manuellement.zip
```

```
--2023-06-11 22:30:31--  http://liris.univ-lyon2.fr/reves/documents/Extrait_Pl@ntLeaveI_segmente_manuellement.zip
Resolving liris.univ-lyon2.fr (liris.univ-lyon2.fr)... 159.84.143.100
Connecting to liris.univ-lyon2.fr (liris.univ-lyon2.fr)|159.84.143.100|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42226533 (40M) [application/zip]
Saving to: '/content/leaf-segmentation/raw-dataset/Extrait_Pl@ntLeaveI_segmente_manuellement.zip'

Extrait_Pl@ntLeaveI 100%[===================>]  40.27M  4.95MB/s    in 10s
```

سلول زیر، یک فایل زیپ را اکسترکت می‌کند و محتویات آن را در دایرکتوری مشخص شده ذخیره می‌کند. Path فایل را داریم یک path برای اکسترکت تعیین می‌کنیم که پارامتر exist_ok=True باعث می‌شود که

اگر دایرکتوری از قبل وجود داشته باشد، خطا ایجاد نشود.

به طور خلاصه، این کد فایل زیپ را اکسترکت کرده و محتویات آن را در دایرکتوری مورد نظر ذخیره می‌کند.

```
zip_file_path = "/content/leaf-segmentation/raw-dataset/Extrait_Pl@ntLeaveI_segmente_manuellement.zip"


destination_directory = "/content/leaf-segmentation/raw-dataset/"

# Create the extract directory if it doesn't exist
os.makedirs(destination_directory, exist_ok=True)

# Extract the contents of the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(destination_directory)

print("Extraction completed successfully.")
```

```
Extraction completed successfully.
```

سلول زیر، پیش پردازش و سازماندهی داده های موجود در دایرکتوری مورد نظر را انجام می‌دهد. به منظور جدا سازی تصاویر و ماسک ها دایرکتوری هایی برای آن ها تعریف شد و در صورتی که قبلا وجود نداشته باشند ایجاد می‌شوند.

در متغیر های image_files و mask_files لیستی از فایل های تصویر و ماسک نگهداری می‌شود و تعداد آن ها را به دست می‌آوریم.

به طور خلاصه، این سلول، فایل های تصویر و ماسک موجود در یک دایرکتوری را به دایرکتوری های مجزا منتقل می‌کند و سایر فایل ها و دایرکتوری ها را از دایرکتوری اصلی حذف می‌کند. سپس، تعداد تصاویر و ماسک ها را چاپ می‌کند.

```python
# Set the paths
raw_dataset_folder = "/content/leaf-segmentation/raw-dataset"
images_folder = os.path.join(raw_dataset_folder, "images")
masks_folder = os.path.join(raw_dataset_folder, "masks")

# Create the "images" and "masks" folders
os.makedirs(images_folder, exist_ok=True)
os.makedirs(masks_folder, exist_ok=True)

# Get the path of the "Extrait_Pl@ntLeaveI_segmente_manuellement" folder
extrait_folder = os.path.join(raw_dataset_folder, "Extrait_Pl@ntLeaveI_segmente_manuellement")

# Get the list of image files
image_files = []
mask_files = []
for file_name in os.listdir(extrait_folder):
    if file_name.endswith("s.jpg"):
        mask_files.append(file_name)
    elif file_name.endswith(".jpg"):
        image_files.append(file_name)

# Move image files to the "images" folder
for image_file in image_files:
    image_src = os.path.join(extrait_folder, image_file)
    image_dest = os.path.join(images_folder, image_file)
    shutil.copy(image_src, image_dest)

# Move mask files to the "masks" folder
```

```python
# Move mask files to the "masks" folder
for mask_file in mask_files:
    mask_src = os.path.join(extrait_folder, mask_file)
    mask_dest = os.path.join(masks_folder, mask_file)
    shutil.copy(mask_src, mask_dest)

print("Copying images and masks completed successfully.")

# Remove all files and folders inside the "raw_dataset" directory, except for "images", "masks", and "Extrait_Pl@ntLeaveI_segmente_manuellement.zip"
for file_name in os.listdir(raw_dataset_folder):
    file_path = os.path.join(raw_dataset_folder, file_name)
    if file_name != "images" and file_name != "masks" and file_name != "Extrait_Pl@ntLeaveI_segmente_manuellement.zip":
        if os.path.isfile(file_path):
            os.remove(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)

print("Unnecessary files and folders removed.")

# Print the number of images and masks
num_images = len(os.listdir(images_folder))
num_masks = len(os.listdir(masks_folder))
print(f"Number of images: {num_images}")
print(f"Number of masks: {num_masks}")
```

```
Copying images and masks completed successfully.
Unnecessary files and folders removed.
Number of images: 233
Number of masks: 233
```

در سلول زیر، اندازه های حداکثر و حداقل تصاویر محاسبه می‌شود.

calculate_max_min_dims(directory) این تابع، بر اساس مسیر داده شده، اندازه های حداکثر و حداقل تصاویر را محاسبه می‌کند و آنها را به صورت یک تاپل برمی‌گرداند.

```
[ ]  def calculate_max_min_dims(directory):
         max_w_dim = 0
         max_h_dim = 0
         min_w_dim = 10000
         min_h_dim = 10000

         # Iterate over the images in the directory
         for image_name in os.listdir(directory):
             image_path = os.path.join(directory, image_name)
             img = Image.open(image_path)
             width, height = img.size
             max_w_dim = max(max_w_dim, width)
             max_h_dim = max(max_h_dim, height)
             min_w_dim = min(min_w_dim, width)
             min_h_dim = min(min_h_dim, height)


         sizes = (max_w_dim, max_h_dim, min_w_dim, min_h_dim)
         return sizes
```

سلول زیر، اندازه های حداکثر و حداقل تصاویر در دایرکتوری داده شده را با تابع سلول بالا محاسبه می‌کند و پرینت می‌گیرد.

```
images_dir = '/content/leaf-segmentation/raw-dataset/images'
print(f'max_w_dim, max_h_dim, min_w_dim, min_h_dim for images are: {calculate_max_min_dims(images_dir)}')
```

```
max_w_dim, max_h_dim, min_w_dim, min_h_dim for images are: (800, 800, 161, 173)
```

سلول زیر، تقسیم بندی مجموعه داده برای آموزش، اعتبارسنجی و آزمون را انجام می‌دهد. به این منظور دایرکتوری های مورد نیاز را ایجاد می‌کند.

تصاویر و ماسک ها با شماره های مشخص شده برای تست جدا می‌شوند(برای من از 3977 تا 4903 بود) سپس بقیه‌ی تصاویر و ماسک ها به طور تصادفی به دو مجموعه train و dev تقسیم شدند و در نهایت، تعداد تصاویر و ماسک ها در هر دایرکتوری چاپ می‌شود.

از مجموعه داده‌ی train برای آموزش مدل شبکه عصبی استفاده می‌شود. شبکه عصبی با استفاده از این مجموعه داده آموزش می‌بیند و وزن ها و پارامترهای خود را تنظیم می‌کند تا بتواند بر روی داده های جدید دقت مناسبی داشته باشد.

از مجموعه داده‌ی dev به عنوان مجموعه اعتبار سنجی برای تنظیم و انتخاب بهترین پارامتر های مدل استفاده می‌شود. با استفاده از این داده ها، مدل با پارامترهای مختلف آموزش می‌بیند و روی مجموعه اعتبارسنجی ارزیابی می‌شود. این کار به ما کمک میکند تا بهترین پارامترها را انتخاب کنیم و از پدیده هایی مانند overfitting جلوگیری کنیم.

از داده های test برای ارزیابی نهایی مدل استفاده می‌شود. از این داده ها معمولاً از دسته بندی مدل جدید در شرایط واقعی و روی داده های جدید استفاده می‌شوند. با ارزیابی مدل بر روی این داده ها، می‌توانیم دقت نهایی و عملکرد مدل را ارزیابی کنیم.

تفاوت اصلی بین داده های test و dev در استفاده از آن ها در فرآیند ارزیابی مدل است. در کل، دسته `dev`برای تنظیم بهتر پارامتر ها و دسته `test` برای ارزیابی نهایی و ارزیابی عملکرد نهایی مدل استفاده می‌شود.

نفاوت داده های dev و train هم در این است که از دسته dev برای ارزیابی مدل بین دوره های آموزش استفاده می‌شود و عملکرد مدل بر روی این دسته داده ها نمایش داده می‌شود.

```python
# Set the paths
leaf_segmentation_folder = "/content/leaf-segmentation"
dataset_folder = os.path.join(leaf_segmentation_folder, "dataset")
raw_dataset_folder = os.path.join(leaf_segmentation_folder, "raw-dataset")
train_folder = os.path.join(dataset_folder, "train")
dev_folder = os.path.join(dataset_folder, "dev")
test_folder = os.path.join(dataset_folder, "test")

# Remove existing train, dev, and test folders if they exist
shutil.rmtree(train_folder, ignore_errors=True)
shutil.rmtree(dev_folder, ignore_errors=True)
shutil.rmtree(test_folder, ignore_errors=True)

# Create the train, dev, and test folders
os.makedirs(train_folder, exist_ok=True)
os.makedirs(dev_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

# Set the paths for train, dev, and test images and masks
train_images_folder = os.path.join(train_folder, "images", "img")
train_masks_folder = os.path.join(train_folder, "masks", "img")
dev_images_folder = os.path.join(dev_folder, "images", "img")
dev_masks_folder = os.path.join(dev_folder, "masks", "img")
test_images_folder = os.path.join(test_folder, "images", "img")
test_masks_folder = os.path.join(test_folder, "masks", "img")

# Create images and masks folders within train, dev, and test directories
```

```
    # Create images and masks folders within train, dev, and test directories
    os.makedirs(train_images_folder, exist_ok=True)
    os.makedirs(train_masks_folder, exist_ok=True)
    os.makedirs(dev_images_folder, exist_ok=True)
    os.makedirs(dev_masks_folder, exist_ok=True)
    os.makedirs(test_images_folder, exist_ok=True)
    os.makedirs(test_masks_folder, exist_ok=True)

    # Copy images and masks to the test set
    test_start_index = 3977
    test_end_index = 4903
    for i in range(test_start_index, test_end_index + 1):
        image_filename = str(i) + ".jpg"
        mask_filename = str(i) + "s.jpg"
        image_src = os.path.join(raw_dataset_folder, "images", image_filename)
        mask_src = os.path.join(raw_dataset_folder, "masks", mask_filename)
        image_dest = os.path.join(test_images_folder, image_filename)
        mask_dest = os.path.join(test_masks_folder, mask_filename)
        if os.path.exists(image_src):
            shutil.copy(image_src, image_dest)
            shutil.copy(mask_src, mask_dest)

    # Obtain the remaining images (excluding the ones copied to the test set)
    remaining_images = []
    for image in os.listdir(os.path.join(raw_dataset_folder, "images")):
        image_filename = image[:-4]
        if int(image_filename) < test_start_index or int(image_filename) > test_end_index:
            remaining_images.append(image)
```

```
# Randomly split the remaining images into train and dev sets
random.shuffle(remaining_images)
num_train = int(0.9 * len(remaining_images))
train_images = remaining_images[:num_train]
dev_images = remaining_images[num_train:]

# Copy train images and masks to the train set
for image in train_images:
    image_src = os.path.join(raw_dataset_folder, "images", image)
    mask_src = os.path.join(raw_dataset_folder, "masks", image[:-4] + "s.jpg")
    image_dest = os.path.join(train_images_folder, image)
    mask_dest = os.path.join(train_masks_folder, image[:-4] + "s.jpg")
    shutil.copy(image_src, image_dest)
    shutil.copy(mask_src, mask_dest)

# Move dev images and masks to the dev set
for image in dev_images:
    image_src = os.path.join(raw_dataset_folder, "images", image)
    mask_src = os.path.join(raw_dataset_folder, "masks", image[:-4] + "s.jpg")
    image_dest = os.path.join(dev_images_folder, image)
    mask_dest = os.path.join(dev_masks_folder, image[:-4] + "s.jpg")
    shutil.copy(image_src, image_dest)
    shutil.copy(mask_src, mask_dest)

print("Images and masks copied to train, dev, and test sets successfully.")
```

```
# Print the number of images and masks in each folder
train_num_images = len(os.listdir(train_images_folder))
train_num_masks = len(os.listdir(train_masks_folder))
dev_num_images = len(os.listdir(dev_images_folder))
dev_num_masks = len(os.listdir(dev_masks_folder))
test_num_images = len(os.listdir(test_images_folder))
test_num_masks = len(os.listdir(test_masks_folder))

print(f"Train set - Number of images: {train_num_images}, Number of masks: {train_num_masks}")
print(f"Dev set - Number of images: {dev_num_images}, Number of masks: {dev_num_masks}")
print(f"Test set - Number of images: {test_num_images}, Number of masks: {test_num_masks}")
```

```
Images and masks copied to train, dev, and test sets successfully.
Train set - Number of images: 177, Number of masks: 177
Dev set - Number of images: 20, Number of masks: 20
Test set - Number of images: 36, Number of masks: 36
```

در سلول زیر تابع اول، تصویر و ماسک را در کنار هم نمایش می‌دهد. برای این منظور، ابتدا یک figure جدید با دو subplots در نظر گرفته می‌شود. و تابع دوم، تصویر اصلی به همراه ماسک واقعی و ماسک پیش بینی شده را در کنار هم نمایش می‌دهد. برای این منظور، ابتدا figure جدید با سه subplots در نظر گرفته شد و title مناسب برای هر کدام در نظر گرفته شد.

```python
def display_image_with_mask(image, mask):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
    ax1.imshow(image)
    ax1.set_title('Image')
    ax1.axis('off')
    ax2.imshow(mask, cmap='gray')
    ax2.set_title('Mask')
    ax2.axis('off')
    plt.show()


def display_image_with_true_and_predicted_masks(image, true_mask, predicted_mask):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
    ax1.imshow(image)
    ax1.set_title('Image')
    ax1.axis('off')
    ax2.imshow(true_mask, cmap='gray')
    ax2.set_title('True Mask')
    ax2.axis('off')
    ax3.imshow(predicted_mask, cmap='gray')
    ax3.set_title('Predicted Mask')
    ax3.axis('off')
    plt.show()
```

در تابع زیر، تصاویر تست را به همراه ماسک های واقعی و پیش بینی شده در کنار هم نمایش می‌دهد. برای این منظور، ابتدا تعداد تصاویر مورد نظر را محاسبه می‌کند و سپس تعداد سطر ها و ستون ها در نمودار را محاسبه می‌کند که در اینجا تعداد ستون ها ثابت است و برابر با ۳ است. سپس یک figure جدید با subplots به تعداد تصاویر ساخته می‌شود.

در ادامه، برای هر تصویر و ماسک متناظر، axes مربوطه را دریافت می‌کند.

سپس تصویر اصلی را در ax1 نمایش می‌دهد و عنوان Image را برای آن تعیین می‌کند. همچنین، ماسک واقعی را در ax2 و ماسک پیشبینی شده را در ax3 نمایش می‌دهد. عنوان True Maskو Predicted Mask به ترتیب برای آنها تعیین می‌شود.

```python
def display_images_with_masks(images, true_masks, predicted_masks, titles=None):
    num_images = len(images)

    # Calculate the number of rows and columns in the plot
    num_rows = num_images
    num_cols = 3

    # Create the plot with subplots
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_images))

    # Iterate over each image and its corresponding masks
    for i in range(num_images):
        image = images[i]
        true_mask = true_masks[i]
        predicted_mask = predicted_masks[i]

        # Get the corresponding axes for the current row
        if num_rows > 1:
            ax1, ax2, ax3 = axes[i]
        else:
            ax1, ax2, ax3 = axes

        # Display the image
        ax1.imshow(image)
        ax1.set_title('Image')

        # Display the true mask
        ax2.imshow(true_mask, cmap='gray')
        ax2.set_title('True Mask')

        # Display the predicted mask
```

```python
        # Display the true mask
        ax2.imshow(true_mask, cmap='gray')
        ax2.set_title('True Mask')

        # Display the predicted mask
        ax3.imshow(predicted_mask, cmap='gray')
        ax3.set_title('Predicted Mask')

        # Turn off the axis labels
        ax1.axis('off')
        ax2.axis('off')
        ax3.axis('off')

        # Set the title for the current row if provided
        if titles is not None:
            ax1.set_title(titles[i])

    # Adjust the spacing between subplots
    plt.tight_layout()

    # Show the plot
    plt.show()
```

تابع زیر، accuracy ماسک های واقعی و پیش بینی شده را محاسبه می‌کند و نمایش می‌دهد. ابتدا یک لیست برای ذخیره دقت تصویر ایجاد می‌شود.

سپس یک نمونه از tf.keras.metrics.Accuracy ایجاد می‌شود. در هر دوره از حلقه، برای هر ماسک واقعی و پیش بینی شده، دقت با استفاده از متدهای update_state ورودی های واقعی و پیش بینی شده را به روزرسانی می‌کند. سپس با استفاده از numpy، مقدار دقت را دریافت می‌کند و به لیست دقت ها اضافه می‌کند.

پس از پایان حلقه، overall accuracy محاسبه می‌شود با استفاده از np.mean(accuracies) که میانگین دقت همه تصاویر است. سپس دقت کلی را چاپ می‌کند.

پس از آن برای رسم نمودار دقت ها، ابتدا یک figure مشخص ایجاد می‌شود. سپس با استفاده از plt.bar، نمودار میله ای دقت برای هر تصویر را نمایش میدهد.

```python
def plot_accuracies(true_masks, preds_masks):
    # Calculate accuracy for each image
    accuracies = []

    # Create an instance of tf.keras.metrics.Accuracy
    accuracy_metric = tf.keras.metrics.Accuracy()
    for i in range(len(true_masks)):
        true_mask = true_masks[i]
        pred_mask = preds_masks[i]

        # Update the metric with the true and predicted labels
        accuracy_metric.update_state(true_mask, pred_mask)
        # Get the accuracy value
        accuracy = accuracy_metric.result().numpy()

        accuracies.append(accuracy)
        print(f"Accuracy for image {i+1}: {round(accuracy*100, 2)}%")

    # Calculate the overall accuracy
    overall_accuracy = np.mean(accuracies)
    print(f"Overall accuracy: {round(overall_accuracy*100, 2)}%")

    # Plotting
    plt.figure(figsize=(8, 6))
    plt.bar(range(1, len(accuracies) + 1), accuracies)
    plt.axhline(overall_accuracy, color='red', linestyle='--', label=f'Overall Accuracy: {round(overall_accuracy*100, 2)}%')
    plt.xlabel('Image')
    plt.ylabel('Accuracy')
    plt.title('Accuracy for Each Image')
    plt.legend()
    plt.show()
```

تابع زیر، یک تصویر و یک ماسک را به عنوان ورودی دریافت می‌کند و تصویری را نشان میدهد که حاوی یک خط قرمز دور برگ است.

ابتدا تبدیل ماسک به سطح خاکستری با np.uint8(mask) را داریم. پس از آن تابع گوسی را برای

هموارسازی اعمال می‌کنیم با Gaussian blur، سپس Laplacian اعمال می‌کنیم برای تشخیص لبه ها و dilation روی لبه ها اعمال می‌کنیم. dilation روی لبه های عکس به منظور تقویت و گسترش لبه ها انجام می‌شود. بعد از آن با استفاده از cv2.cvtColor تصویر RGB را از تصویر سطح خاکستری خطوط ترسیم شده ایجاد می‌کنیم و سپس برای پیکسل های غیر صفر خطوط ترسیم شده رنگ قرمز) را تخصیص می‌دهیم. سپس تصویر dilated_edges_rgb را به نوع داده مانند تصویر اصلی تبدیل می‌کنیم و آن را به تصویر اصلی اضافه می‌کنیم و در آخر برگ با مرز ها و لبه های قرمز را نمایش می‌دهیم.

```python
def display_masked_image_object_detection_style(image, mask):
    '''This function draws a red line around the leaf'''
    # Convert the mask to grayscale
    mask = np.uint8(mask)

    # Apply Gaussian blur to the mask to get smooth edges
    blurred = cv2.GaussianBlur(mask, (5, 5), 0)

    # Apply the Laplacian edge detection on the blurred mask
    edges = cv2.Laplacian(blurred, cv2.CV_8U)

    # Perform dilation on the edges
    dilated_edges = cv2.dilate(edges, None, iterations=1)

    # Create an RGB image from the dilated edges
    dilated_edges_rgb = cv2.cvtColor(dilated_edges, cv2.COLOR_GRAY2RGB)
    dilated_edges_rgb[dilated_edges > 0] = (255, 0, 0)

    # Convert the dilated_edges_rgb to the same data type as image
    dilated_edges_rgb = dilated_edges_rgb.astype(image.dtype)

    # Add the dilated_edges_rgb image to the original image
    output_image = cv2.addWeighted(image, 1, dilated_edges_rgb, 1, 0)

    # Display the masked image
    plt.imshow(output_image)
    plt.axis('off')
    plt.show()
```

در سلول زیر، انواع تنظیمات و هایپرپارامتر هایی را که برای آموزش مدل استفاده می‌شوند، از جمله مسیر داده، اندازه ورودی، اندازه دسته، تعداد تکرارها، نرخ یادگیری و سایر پارامترهای مربوط به استفاده از Callback ها مشخص شد.

ابتدا مسیر داده های آموزش، اعتبارسنجی و تست را مشخص کردیم. این مسیر ها به پوشه هایی اشاره می‌کنند که داده های متناظر در آن قرار دارند. سپس اندازه تصاویر ورودی را مشخص کردیم که 256 در 256 پیکسل بود. BATCH_SIZE را 8 قرار دادیم که یعنی تعداد نمونه هایی که هر بار در طول آموزش به مدل داده

می‌شود. در اینجا، هر دسته شامل 8 تصویر است. SEED = 42 قرار دادیم که برای تولید اعداد تصادفی تنظیم می‌کند. EPOCHS = 250 که تعداد تکرار های کامل آموزش مدل را مشخص می‌کند و هر epoch شامل یک بار گذر کامل از داده های آموزش است. STEPS_PER_EPOCH = 23 که تعداد steps در هر تکرار را مشخص می‌کند. LR = 0.0002 قرار دادیم این مقدار نشان میدهد که چقدر یادگیری پارامترهای مدل در هر مرحله بهبود یابد.

HIGHEST_ACC_FOR_CALL_BACK = 0.98 قرار دادیم که بهترین accuracy مدل در هر آموزش را نشان می‌دهد .

```
TRAIN_PATH = '/content/leaf-segmentation/dataset/train'
DEV_PATH = '/content/leaf-segmentation/dataset/dev'
TEST_PATH = '/content/leaf-segmentation/dataset/test'
INPUT_SIZE = (256, 256)
BATCH_SIZE = 8
SEED = 42

EPOCHS = 250
STEPS_PER_EPOCH = 23
LR = 0.0002
HIGHEST_ACC_FOR_CALL_BACK = 0.98
```

در سلول زیر، data generator برای تبدیل تصاویر و ماسک ها به فرمت مناسب برای آموزش، اعتبارسنجی و تست مدل آماده شدند. علاوه بر آن، با استفاده از پارامترهای اضافی مانند horizontal_flip و vertical_flip، تصاویر و ماسک ها می‌توانند تغییرات جانبی مثل برگرداندن افقی یا عمودی را دریافت کنند که برای افزایش تنوع داده و بهبود عملکرد مدل مفید هست. همچنین، مقیاس دهی مقادیر پیکسل با استفاده از rescale به نرمال سازی داده ها کمک می‌کند تا مقیاس مقادیر پیکسل بین 0 و 1 قرار بگیرد.

```python
# Create the ImageDataGenerator for train, dev, and test images and masks
train_image_data_gen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    horizontal_flip=True,
    vertical_flip=True
)
train_mask_data_gen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    horizontal_flip=True,
    vertical_flip=True
)
dev_image_data_gen = ImageDataGenerator(rescale=1.0 / 255.0)
dev_mask_data_gen = ImageDataGenerator(rescale=1.0 / 255.0)
test_image_data_gen = ImageDataGenerator(rescale=1.0 / 255.0)
test_mask_data_gen = ImageDataGenerator(rescale=1.0 / 255.0)
```

در ادامه، تنظیم تولیدکننده های داده برای آموزش، اعتبار سنجی و آزمون مدل است. این تولیدکننده ها، داده‌های تصویر و ماسک را از پوشه های مربوطه می‌خوانند و آنها را به صورت دسته هایی با اندازه و تنظیمات مشخص تبدیل می‌کنند. این دسته های تولید شده می‌توانند به عنوان ورودی به مدل آموزش داده شوند.

برای داده های train، shuffle تنظیم می‌شود تا برای تصادفی کردن ترتیب داده ها و ماسک هاgrayscale تبدیل می‌شوند.

برای داده های تست نیازی به shuffle کردن نیست.

در ادامه ماسک های train و dev را باینری می‌کنیم. با تابع preprocess_mask به طوری که مقادیر بزرگ تر از 0 را یک می‌کنیم سپس به طور بولی آن را not می‌کنیم که یعنی محدوده مشکی و سفید آن برعکس می‌شوند.

سپس از توابع zip به منظور ایجاد ترکیبی از تصاویر و ماسک ها به صورت tuple استفاده می‌شود. هر tuple شامل یک تصویر و ماسک متناظر است.

```python
# Use the target size and class mode in the ImageDataGenerator for train images and masks
train_image_generator = train_image_data_gen.flow_from_directory(
    directory=os.path.join(TRAIN_PATH, 'images'),
    target_size=INPUT_SIZE,
    batch_size=BATCH_SIZE,
    class_mode=None,
    shuffle=True,
    seed=42
)
train_mask_generator = train_mask_data_gen.flow_from_directory(
    directory=os.path.join(TRAIN_PATH, 'masks'),
    target_size=INPUT_SIZE,
    batch_size=BATCH_SIZE,
    class_mode=None,
    color_mode='grayscale',
    shuffle=True,
    seed=42
)

# Use the target size and class mode in the ImageDataGenerator for dev images and masks
dev_image_generator = dev_image_data_gen.flow_from_directory(
    directory=os.path.join(DEV_PATH, 'images'),
    target_size=INPUT_SIZE,
    batch_size=BATCH_SIZE,
    class_mode=None,
    shuffle=True,
    seed=42
)
dev_mask_generator = dev_mask_data_gen.flow_from_directory(
```

```python
dev_mask_generator = dev_mask_data_gen.flow_from_directory(
    directory=os.path.join(DEV_PATH, 'masks'),
    target_size=INPUT_SIZE,
    batch_size=BATCH_SIZE,
    class_mode=None,
    color_mode='grayscale',
    shuffle=True,
    seed=42
)

# Use the target size and class mode in the ImageDataGenerator for test images and masks
test_image_generator = test_image_data_gen.flow_from_directory(
    directory=os.path.join(TEST_PATH, 'images'),
    target_size=INPUT_SIZE,
    batch_size=36,
    class_mode=None,
    shuffle=False  # No need to shuffle test data
)
test_mask_generator = test_mask_data_gen.flow_from_directory(
    directory=os.path.join(TEST_PATH, 'masks'),
    target_size=INPUT_SIZE,
    batch_size=36,
    class_mode=None,
    color_mode='grayscale',
    shuffle=False  # No need to shuffle test data
)

def preprocess_mask(mask):
    mask[mask > 0] = 1
    mask = np.logical_not(mask).astype(mask.dtype)
    return mask
```

```python
def preprocess_mask(mask):
    mask[mask > 0] = 1
    mask = np.logical_not(mask).astype(mask.dtype)
    return mask

train_mask_generator = (preprocess_mask(mask) for mask in train_mask_generator)
dev_mask_generator = (preprocess_mask(mask) for mask in dev_mask_generator)

# Create the train, dev, and test generators
train_generator = zip(train_image_generator, train_mask_generator)
dev_generator = zip(dev_image_generator, dev_mask_generator)
test_generator = zip(test_image_generator, test_mask_generator)
```

```
Found 177 images belonging to 1 classes.
Found 177 images belonging to 1 classes.
Found 20 images belonging to 1 classes.
Found 20 images belonging to 1 classes.
Found 36 images belonging to 1 classes.
Found 36 images belonging to 1 classes.
```

در سلول زیر، ابتدا یک دسته تصادفی از تصاویر و ماسک ها از train_generator به دست می‌آید. سپس یک شماره اندیس تصادفی از تصاویر و ماسک ها در این دسته را انتخاب می‌کند. در نهایت، تصویر و ماسک متناظر با این شماره اندیس را از دسته تصادفی بدست می‌آورد.

```python
# Get a random batch of images and masks from the train generator
images, masks = next(train_generator)

# Choose a random index from the batch
random_index = np.random.randint(0, len(images))

# Get the random image and its mask
image = images[random_index]
mask = masks[random_index]
```

در سلول زیر، دو متغیر image و mask به ترتیب اندازه تصویر و ماسک را نشان می‌دهند.

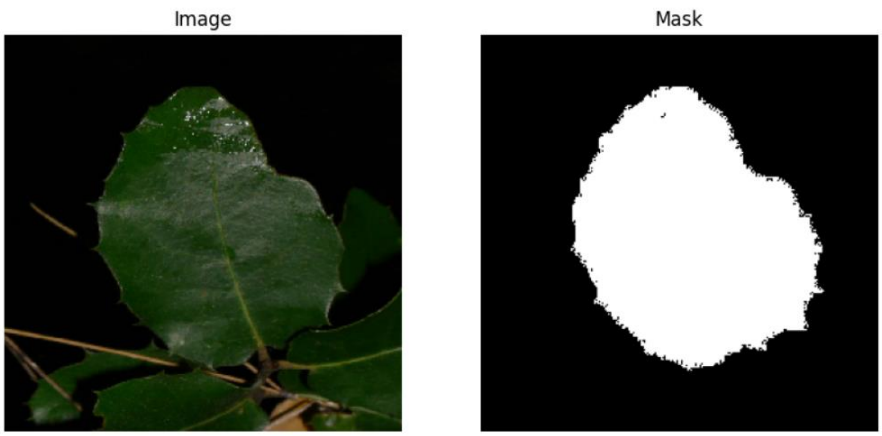
خروجی این کد به صورت (256,256,3)این بدان معنی است که تصویر دارای ابعاد 256 در 256 پیکسل و با 3 کانال رنگ RGB است، در حالی که ماسک ابعاد 256 در 256 پیکسل را دارد و تنها دارای یک کانال (سطح خاکستری) است.

```python
[ ] image.shape, mask.shape
```

```
((256, 256, 3), (256, 256, 1))
```

در سلول زیر با تابعی که پیشتر توضیح دادیم تصویر و ماسک باینری معکوس شده ی متناظر با آن را نشان می‌دهیم.

```
display_image_with_mask(image, mask)
```



در سلول زیر با تابعی که پیشتر توضیح دادیم لبه های برگ را با رنگ قرمز مشخص کردیم.

```
display_masked_image_object_detection_style(image, mask)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

در این پروژه از معماری U-net استفاده کردیم. معماری U-Net یک Convolutional Neural Network برای مسائل دسته بندی تصویر و تشخیص الگو است. این معماری از ساختاری با دو بخش اصلی تشکیل شده است Encoder و Decoder. در این معماری، اطلاعات تصویر از طریق لایه های کانولوشنال به سمت پایین جریان می‌یابد(Encoder) و سپس از طریق لایه های Transpose به سمت بالا جریان می‌یابد(Decoder). این ساختار نیرومند به شبکه اجازه می‌دهد تا اطلاعات دقیق مکانی تصویر را در فرآیند تشخیص الگو حفظ کند.

در سلول زیر، مدل U-Net ار ایجاد می‌کنیم. ابتدا ابعاد ورودی را برای لایه ی ورودی مشخص می‌کنیم. در ساختار Encoder اعمال لایه های کانولوشن با 64 فیلتر و 128 فیلتر و 512 و 1024 فیلتر 3 در 3، روی لایه های قبلی با padding های متناظر و تابع فعالسازی ReLU را داریم و همچین در لایه های MaxPooling را در ابعاد 2 در 2 اعمال کردیم.

در بخش Decoder لایه ها را concatenate می‌کنیم در هر مرحله تا آخر. عملیات upsample را در جهت عکس قبلی ادامه می‌دهیم.

به طور خلاصه، در سلول زیر مدل U-Net را تعریف کردیم که شامل لایه های کانولوشنال، لایه های حذف نمونه ها، لایه های upsample و لایه های ادغام است. خروجی نهایی آن با استفاده از تابع فعالسازی Sigmoid اعمال می‌شود.

تابع سیگموئید باعث تنظیم خروجی لایه های عصبی بین 0 و 1 می‌شود. وقتی ورودی تابع به مقدار بزرگی میل کند، خروجی نزدیک به 1 می‌شود، و وقتی ورودی به مقدار کوچکی میل کند، خروجی نزدیک به 0 می‌شود. این ویژگی باعث میشود تا تابع سیگموئید مناسب برای مسائل دسته‌بندی دو کلاسه و تبدیل خروجی به احتمال بین 0 و 1 باشد.

```python
# Define the U-Net model
def unet(input_size):
    # Input layer
    inputs = Input(shape=input_size)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4)
    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(conv5)

    # Decoder
    up6 = UpSampling2D(size=(2, 2))(conv5)
    up6 = Conv2D(512, 2, activation='relu', padding='same')(up6)
    merge6 = concatenate([conv4, up6], axis=3)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(merge6)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)
```

```
    up7 = UpSampling2D(size=(2, 2))(conv6)
    up7 = Conv2D(256, 2, activation='relu', padding='same')(up7)
    merge7 = concatenate([conv3, up7], axis=3)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(merge7)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

    up8 = UpSampling2D(size=(2, 2))(conv7)
    up8 = Conv2D(128, 2, activation='relu', padding='same')(up8)
    merge8 = concatenate([conv2, up8], axis=3)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(merge8)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

    up9 = UpSampling2D(size=(2, 2))(conv8)
    up9 = Conv2D(64, 2, activation='relu', padding='same')(up9)
    merge9 = concatenate([conv1, up9], axis=3)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(merge9)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)

    # Output layer
    conv10 = Conv2D(1, 1, activation='sigmoid')(conv9)

    # Create the model
    model = Model(inputs=inputs, outputs=conv10)

    return model
```

سلول زیر، مدل U-Net را ایجاد میکند با ورودی به اندازه INPUT_SIZE به همراه ۳ کانال برای تصاویر رنگی. تابع unet از قبل تعریف شده است و با دادن اندازه ورودی به آن، مدل U-Net را ساخته و بازگردانده میشود.

```
model = unet((INPUT_SIZE + (3,)))
```

در ادامه، تابع model.summary در Keras استفاده می‌شود تا جمع بندی جزئیات معماری مدل را نمایش دهد. خروجی این تابع شامل اطلاعات مربوط به هر لایه از مدل است و ترتیب اعمال آن ها و خروجی هر یک از این لایه ها است. در کل، مدل U-Net ما شامل ۲۲ لایه Conv2D، ۶ لایه UpSampling2D و ۴ لایه Concatenate است. تعداد کل پارامترهای قابل آموزش در مدل ۳۱،۰۳۱،۷۴۵ است.

```
model.summary()
```

خروجی سلول و شرح مختصر لایه ها:

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 256, 256, 64  1792        ['input_1[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 256, 256, 64  36928       ['conv2d[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 128, 128, 64  0           ['conv2d_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 128, 128, 12  73856       ['max_pooling2d[0][0]']
                                8)

 conv2d_3 (Conv2D)              (None, 128, 128, 12  147584      ['conv2d_2[0][0]']
                                8)

 max_pooling2d_1 (MaxPooling2D)  (None, 64, 64, 128)  0          ['conv2d_3[0][0]']

 conv2d_4 (Conv2D)              (None, 64, 64, 256)  295168      ['max_pooling2d_1[0][0]']

 conv2d_5 (Conv2D)              (None, 64, 64, 256)  590080      ['conv2d_4[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 32, 32, 256)  0          ['conv2d_5[0][0]']

 conv2d_6 (Conv2D)              (None, 32, 32, 512)  1180160     ['max_pooling2d_2[0][0]']


 conv2d_6 (Conv2D)              (None, 32, 32, 512)  1180160     ['max_pooling2d_2[0][0]']

 conv2d_7 (Conv2D)              (None, 32, 32, 512)  2359808     ['conv2d_6[0][0]']

 max_pooling2d_3 (MaxPooling2D)  (None, 16, 16, 512)  0          ['conv2d_7[0][0]']

 conv2d_8 (Conv2D)              (None, 16, 16, 1024  4719616     ['max_pooling2d_3[0][0]']
                                )

 conv2d_9 (Conv2D)              (None, 16, 16, 1024  9438208     ['conv2d_8[0][0]']
                                )

 up_sampling2d (UpSampling2D)   (None, 32, 32, 1024  0           ['conv2d_9[0][0]']
                                )

 conv2d_10 (Conv2D)             (None, 32, 32, 512)  2097664     ['up_sampling2d[0][0]']

 concatenate (Concatenate)      (None, 32, 32, 1024  0           ['conv2d_7[0][0]',
                                )                                 'conv2d_10[0][0]']

 conv2d_11 (Conv2D)             (None, 32, 32, 512)  4719104     ['concatenate[0][0]']

 conv2d_12 (Conv2D)             (None, 32, 32, 512)  2359808     ['conv2d_11[0][0]']

 up_sampling2d_1 (UpSampling2D)  (None, 64, 64, 512)  0          ['conv2d_12[0][0]']

 conv2d_13 (Conv2D)             (None, 64, 64, 256)  524544      ['up_sampling2d_1[0][0]']

 concatenate_1 (Concatenate)    (None, 64, 64, 512)  0           ['conv2d_5[0][0]',
                                                                  'conv2d_13[0][0]']

 conv2d_14 (Conv2D)             (None, 64, 64, 256)  1179904     ['concatenate_1[0][0]']
```

```
conv2d_14 (Conv2D)              (None, 64, 64, 256)  1179904     ['concatenate_1[0][0]']

conv2d_15 (Conv2D)              (None, 64, 64, 256)  590080      ['conv2d_14[0][0]']

up_sampling2d_2 (UpSampling2D)  (None, 128, 128, 25  0           ['conv2d_15[0][0]']
                                6)

conv2d_16 (Conv2D)              (None, 128, 128, 12  131200      ['up_sampling2d_2[0][0]']
                                8)

concatenate_2 (Concatenate)     (None, 128, 128, 25  0           ['conv2d_3[0][0]',
                                6)                                'conv2d_16[0][0]']

conv2d_17 (Conv2D)              (None, 128, 128, 12  295040      ['concatenate_2[0][0]']
                                8)

conv2d_18 (Conv2D)              (None, 128, 128, 12  147584      ['conv2d_17[0][0]']
                                8)

up_sampling2d_3 (UpSampling2D)  (None, 256, 256, 12  0           ['conv2d_18[0][0]']
                                8)

conv2d_19 (Conv2D)              (None, 256, 256, 64  32832       ['up_sampling2d_3[0][0]']
                                )

concatenate_3 (Concatenate)     (None, 256, 256, 12  0           ['conv2d_1[0][0]',
                                8)                                'conv2d_19[0][0]']

conv2d_20 (Conv2D)              (None, 256, 256, 64  73792       ['concatenate_3[0][0]']
                                )

conv2d_21 (Conv2D)              (None, 256, 256, 64  36928       ['conv2d_20[0][0]']
                                )

conv2d_22 (Conv2D)              (None, 256, 256, 1)  65          ['conv2d_21[0][0]']

==================================================================================================
Total params: 31,031,745
Trainable params: 31,031,745
Non-trainable params: 0
__________________________________________________________________________________________________
```

در سلول زیر، مدل با استفاده از بهینه ساز Adam، تابع هزینه binary_crossentropy و معیار موثر بودن را بر حسب accuracy کامپایل می‌شود.

binary_crossentropy یک معیار میزان خطا یا اشتباه برای مسائل دسته بندی دودویی است. این معیار برای مقایسه بین یک پیش بینی دسته بندی بین دو کلاس (مثلاً مثبت و منفی) و برچسب واقعی استفاده می‌شود. در حالت دسته بندی دودویی، برای هر نمونه داده، احتمالی بین 0 و 1 به عنوان خروجی مدل تولید میشود که نشان دهنده احتمال تعلق آن نمونه به یک کلاس (مثلاً کلاس مثبت) است. مقدار واقعی برچسب برای آن نمونه نیز 0 یا 1 است. برای محاسبه خطا یا اشتباه بین پیش بینی و برچسب واقعی، از binary_crossentropy استفاده می‌شود.

Adam در هر مرحله از آموزش مقادیر میانگین گرادیان و مقادیر مومنتوم را محاسبه می‌کند و از این دو به منظور بهبود سرعت و کیفیت بهینه سازی استفاده می‌کند. مقدار learning rate در Adam به صورت خودکار تطبیق می‌یابد و مقدار اولیه‌ی آن را برابر با هایپر پارامتری که تعریف کرده بودیم پیشتر قرار دادیم.

```python
optimizer = Adam(learning_rate=LR)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
```

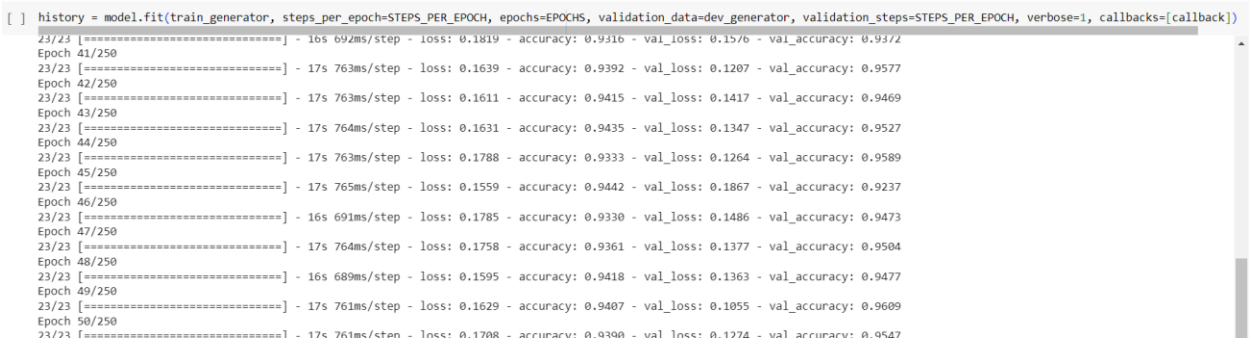
در سلول زیر، یک کلاس با نام AccuracyCallback تعریف شده که یک زیرکلاس از کلاس Callback در کتابخانه Keras است. این کلاس، یک عملکرد خاص بهینه سازی را برای استفاده در آموزش مدل تعریف می‌کند. در این کلاس، متد on_epoch_end تعریف شده است که در پایان هر اپاک آموزش فراخوانی می‌شود. این متد بررسی می‌کند که آیا دقت آموزش برابر یا بیشتر از یک مقدار ثابت به نام HIGHEST_ACC_FOR_CALL_BACK (پیشتر در هایپر پارامتر ها تعریف شد) است یا خیر. اگر دقت برابر یا بیشتر از این مقدار باشد، پیامی چاپ میشود که آموزش با دقت بالاتر از یک درصد به پایان رسیده است. سپس، به کمک ویژگی stop_training مدل، آموزش متوقف می‌شود.

```
class AccuracyCallback(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('accuracy') >= HIGHEST_ACC_FOR_CALL_BACK:
            print(f"\nTraining accuracy reached {HIGHEST_ACC_FOR_CALL_BACK}%!")
            self.model.stop_training = True
callback = AccuracyCallback()
```

در ادامه، آموزش مدل انجام می‌شود. تابع در Keras برای آموزش مدل با داده های آموزش و اعتبار سنجی استفاده می‌شود.

در این کد، متد fit بر روی مدل model فراخوانی می‌شود. پارامترهای ورودی آن در سلول های پیشین تا به اینجا تعریف شده و به این متد پاس داده می‌شود:

نتیجه اجرای تابع fit در متغیر history ذخیره می‌شود که شامل اطلاعاتی مانند مقادیر تابع هزینه و معیارهای عملکرد (مانند دقت) در طول آموزش و اعتبار سنجی است.



سلول زیر، برای اتصال به سیستم فایل Google Drive در محیط Google Colab استفاده می‌شود. با استفاده از این کد، میتوانید به درایو Google خود دسترسی پیدا کنید و فایلها و دادههای موجود در آن را استفاده کنید.

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

```
Mounted at /content/drive
```

سلول زیر برای ذخیره کردن مدل آموزش دیده شده در یک فایل با فرمت H5 استفاده می‌شود. مدل آموزش دیده شده توسط model.saveدر مسیر مشخص شده (`content/drive/MyDrive/leaf-segmentation/model.h5`) ذخیره می‌شود و می‌توانید از آن در زمان های بعدی استفاده کنید چوت train کردن مدل زمان بر هست و داشتن وزن های سیو شده برای ران کردن های بعدی مفید و بهینه تر است.

```
model.save('/content/drive/MyDrive/leaf-segmentation/model.h5')
```

در سلول زیر از Google Drive وزن ها دانلود می‌شوند(بعد از اولین بار اجرا نیازی به اجرای 3 سلول قبل نیست و مستقیما از همین سلول با لینک درایو وزن ها دانلود می‌شود)

و پس از آن وزن ها دانلود شده را در سلول بعدی load می‌کنیم.

```
[ ] !gdown --id 1-3JVZVO2E-KiO0YXsWkiRpXV8nLv1aCZ

/usr/local/lib/python3.10/dist-packages/gdown/cli.py:121: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymore to
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1-3JVZVO2E-KiO0YXsWkiRpXV8nLv1aCZ
To: /content/model.h5
100% 373M/373M [00:04<00:00, 75.3MB/s]

[ ] model.load_weights('/content/model.h5')
```

در سلول زیر برای بهتر شدن نتایج عملیات مورفولوژی روی ماسک های پیش بینی شده اعمال می‌کنیم.

عملیات مورفولوژی برای بهبود ماسکهای باینری استفاده میشود و شامل دو مرحله اصلی است: Opening و Closing.

عملیات Morphological Opening ابتدا اجرا می‌شود. این کار با کم کردن اجزای ریز و اضافی و تمیز کردن ماسک، میتواند نویز های کوچک را حذف کند و اجزای مهم تر را حفظ کند.

سپس عملیات Morphological Closing انجام می‌شود. با انجام این عملیات، تلاش می‌شود جزئیات کوچکی که در عملیات Opening از بین رفته بودند، بازیابی شوند و ماسک بهبود یافته نقاط ناهموار را صاف‌تر نمایش دهد.

سپس با استفاده از عملیات پرکردنfill ، مناطقی که از دست رفته اند یا تخریب شده اند بهبود یافته و تکمیل می‌شوند. این کار باعث ایجاد حاشیه های یکپارچه تر و بهبود شکل ماسک می‌شود.

در نهایت، ماسک های بهبود یافته با ابعاد مشابه ماسک های اولیه برگردانده می‌شوند. این ماسک های بهبود یافته میتوانند دقت و کیفیت نتایج نهایی را افزایش دهند، به خصوص در مواردی که مدل برای شناسایی جزئیات ریز به مشکل میخورد یا نویز هایی در ماسک ها وجود دارد.

در کد تابع apply_morphological_operations این کار را انجام می‌دهد. ورودی این تابع ماسک های پیش بینی شده است. که یک آرایه ی چهار بعدی است با ابعاد (n, height, width, 1) که n تعداد ماسک های پیش بینی شده است.

```python
def apply_morphological_operations(preds_masks):
    n = preds_masks.shape[0]
    output_masks = np.zeros_like(preds_masks)

    for i in range(n):
        pred_mask = preds_masks[i, :, :, 0]

        # Convert mask to uint8
        pred_mask = np.uint8(pred_mask)


        # Apply morphological open on the mask
        kernel = np.ones((7, 7), dtype=np.uint8)
        opened_mask = cv2.morphologyEx(pred_mask, cv2.MORPH_OPEN, kernel=kernel)
        closed_mask = cv2.morphologyEx(opened_mask, cv2.MORPH_CLOSE, kernel=kernel)


        # Fill the opened mask
        filled_mask = closed_mask.copy()
        height, width = filled_mask.shape[:2]
        seed_point = (0, 0)  # Seed point for flood fill
        cv2.floodFill(filled_mask, None, seed_point, 255)

        # Set all the pixels to 1 where the filled mask has a value of 255
        filled_mask = np.where(filled_mask == 255, 1, 0)

        # Expand dimensions to match the original shape
        pred_mask_improved = np.expand_dims(filled_mask, axis=-1)

        output_masks[i] = pred_mask_improved

    return output_masks
```

در سلول زیر، برای استفاده از نتایج پیشبینی، یک آستانه threshold مشخص می‌شود(در اینجا 0.5). سپس مقادیر پیشبینی شده درpreds_masks با آستانه مقایسه می‌شوند. اگر یک مقدار بزرگتر یا مساوی آستانه باشد، به آن مقدار 1 اختصاص داده می‌شود، و در غیر این صورت به آن مقدار 0 اختصاص داده می‌شود. به این ترتیب، مقادیر پیشبینی شده به صورت دودویی (بین 0 و 1) تبدیل میشوند، که به عنوان ماسک های پیش‌بینی استفاده می‌شوند.

```
# Get a batch of validation images and masks
test_images, test_masks = next(test_generator)

# Make predictions
preds_masks = model.predict(test_images)

# Threshold the predictions
threshold = 0.5
preds_masks[preds_masks >= threshold] = 1
preds_masks[preds_masks < threshold] = 0
```

```
2/2 [==============================] - 14s 8ms/step
```

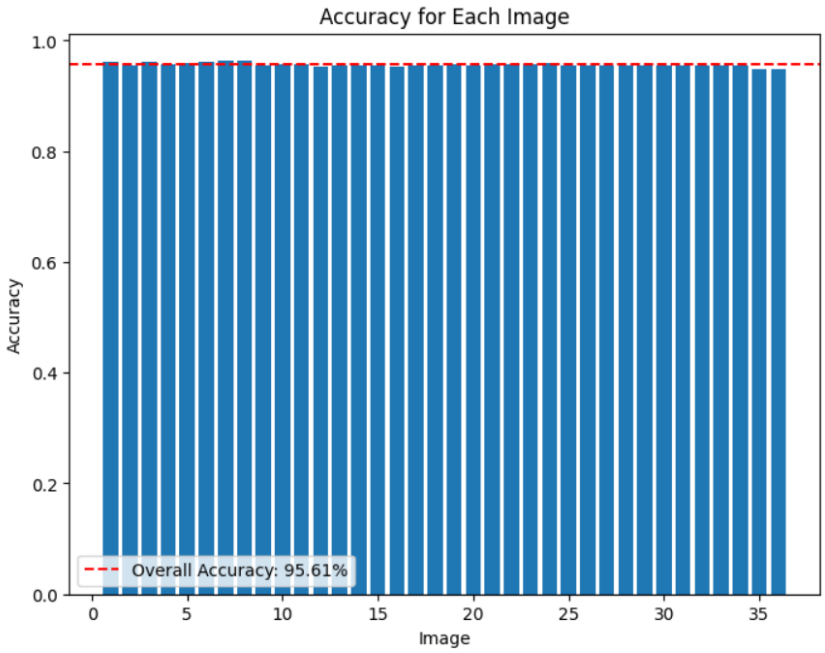
در مرحله بعد با اعمال تابعی که برای عملیات مورفولوژی بود روی ماسک های پیش بینی شده نتایج را بهتر می‌کنیم.

```
[ ]  preds_masks_improved = apply_morphological_operations(preds_masks)
```

و در سلول زیر دقت را بر حسب نمودار میله‌ای نمایش می‌دهیم.(این تابع پیشتر تعریف شده بود و از آن اینجا استفاده می‌کنیم)

ابتدا دقت برای هر تصویر را رسم می‌کنیم و دقت کل با خطوط قرمز روی نمودار نشان داده شده است که

95.61٪ شده است.



در سلول زیر فهرستی از نامهای فایلهای موجود در مسیر تست را دریافت می‌کنیم و آن ها را مرتب می‌کنیم و در متغیر file_names ذخیره می‌کنیم.

```
file_names = os.listdir('/content/leaf-segmentation/dataset/test/images/img')
file_names = sorted(file_names)
```

در ادامه تصاویر اصلی برای تستو ماسک های متناظر با آن ها و ماسک های پیش بینی شده برای آن را نشان می‌دهیم که تابع این هم پیشتر تعریف شده بود.

چند تا از خروجی ها را در ادامه در این گزارش نمایش می‌دهیم.

3977.jpg

True Mask

Predicted Mask

4015.jpg

True Mask

Predicted Mask

4120.jpg

True Mask

Predicted Mask

4131.jpg

True Mask

Predicted Mask

4131.jpg

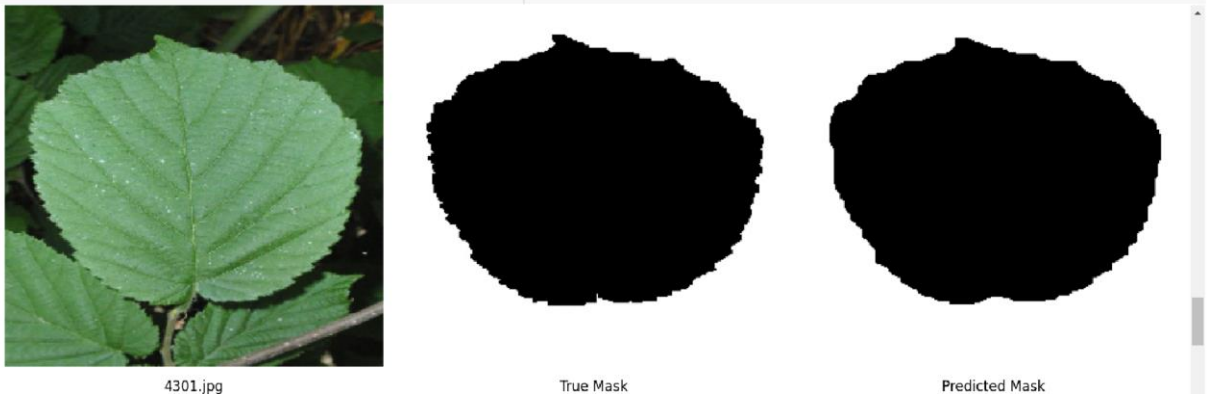
True Mask

Predicted Mask

4144.jpg

True Mask

Predicted Mask



4301.jpg

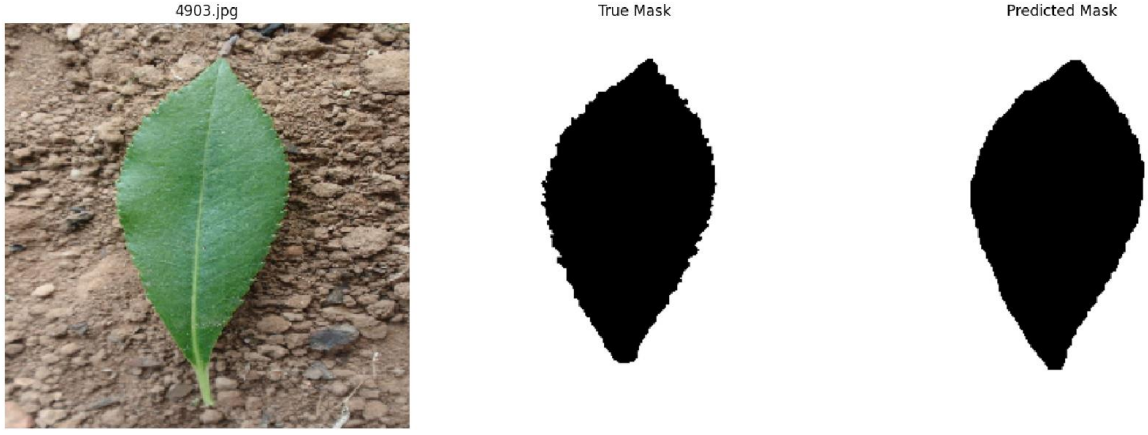
True Mask

Predicted Mask

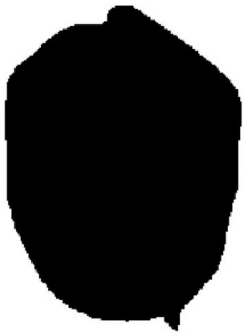
4903.jpg

True Mask

Predicted Mask



4899.jpg



True Mask

Predicted Mask

4888.jpg

True Mask

Predicted Mask

4835.jpg

True Mask

Predicted Mask

4635.jpg

True Mask

Predicted Mask

4635.jpg

True Mask

Predicted Mask