Image segmentation guide for Python

The MediaPipe Image Segmenter task lets you divide images into regions based on predefined categories for applying visual effects such as background blurring. These instructions show you how to use the Image Segmenter with the Python language. For more information about the capabilities, models, and configuration options of this task, see the Overview (https://ai.google.dev/edge/mediapipe/solutions/vision/image_segmenter/index).

Code example

The example code for Image Segmenter provides a complete implementation of this task in Python for your reference. This code helps you test this task and get started on building your own image segmenter application. You can view, run, and edit the Image Segmenter example code (https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/image_segmentation/python/image_segmentation.ipynb) using just your web browser.

Setup

This section describes key steps for setting up your development environment and code projects specifically to use Image Segmenter. For general information on setting up your development environment for using MediaPipe tasks, including platform version requirements, see the Setup guide for Python (/mediapipe/solutions/setup_python). You can review the source code for this example on GitHub (https://github.com/google-ai-edge/mediapipe-samples/tree/main/examples/image_segmentation/python)

Attention: This MediaPipe Solutions Preview is an early release. Learn more (/edge/mediapipe/solutions/about#notice).

Packages

The MediaPipe Image Segmenter task requires the `mediapipe` package. You can install the required dependencies with the following command:

```
$ python -m pip install mediapipe
```

Imports

Import the following classes to access the Image Segmenter task functions:

```
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
```

Model

The MediaPipe Image Segmenter task requires a trained model that is compatible with this task. For more information on available trained models for Image Segmenter, see the task overview Models (https://ai.google.dev/edge/mediapipe/solutions/vision/image_segmenter/index#models) section.

Select and download the model, and then store it within your project directory:

```
model_path = '/absolute/path/to/model.tflite'
```

Specify the path of the model within the Model Name parameter, as shown below:

```
base_options = BaseOptions(model_asset_path=model_path)
```

Create the task

The MediaPipe Image Segmenter task uses the `create_from_options` function to set up the task. The `create_from_options` function accepts values for configuration options to handle. For more information on task configuration, see Configuration options (#configuration_options).

These samples also show the variations of the task construction for images, video files, and live video streams.

```
BaseOptions = mp.tasks.BaseOptions
ImageSegmenter = mp.tasks.vision.ImageSegmenter
ImageSegmenterOptions = mp.tasks.vision.ImageSegmenterOptions
VisionRunningMode = mp.tasks.vision.RunningMode

# Create a image segmenter instance with the image mode:
options = ImageSegmenterOptions(
    base_options=BaseOptions(model_asset_path='/path/to/model.task'),
    running_mode=VisionRunningMode.IMAGE,
    output_category_mask=True)
with ImageSegmenter.create_from_options(options) as segmenter:
```

Note: If you use the live stream mode, you'll need to register a result listener when creating the task. The listener is called whenever the task has finished processing a video frame with the detection result and the input image as parameters.

Note: If you use the video mode or live stream mode, Image Segmenter uses tracking to avoid triggering palm detection model on every frame, and this helps to reduce the latency of Image Segmenter.

Configuration options

This task has the following configuration options for Python applications:

Option Name	Description	Value Range	Default Value
`running_mode`	Sets the running mode for the task. There are three modes: IMAGE: The mode for single image inputs. VIDEO: The mode for decoded frames of a video. LIVE_STREAM: The mode for a livestream of input data, such as from a camera. In this mode, resultListener must be called to set up a listener to receive results asynchronously.	{`IMAGE`, `VIDEO`, `LIVE_STREAM`}	`IMAGE`
`output_category_mask`	If set to `True`, the output includes a segmentation mask as a uint8 image, where each pixel value indicates the winning category value.	{`True`, `False`}	`False`
`output_confidence_`	If set to `True`, the output includes a segmentation mask as a float value image, where each float value represents the confidence score map of the category.	{`True`, `False`}	`True`

Option Name	Description	Value Range	Default Value
`masks`			
`display_names_locale`	Sets the language of labels to use for display names provided in the metadata of the task's model, if available. Default is `en` for English. You can add localized labels to the metadata of a custom model using the TensorFlow Lite Metadata Writer API (https://www.tensorflow.org/lite/models/convert/metadata_writer_tutorial)	Locale code	en
`result_callback`	Sets the result listener to receive the segmentation results asynchronously when the image segmenter is in the `LIVE_STREAM` mode. Can only be used when running mode is set to `LIVE_STREAM`	N/A	N/A

Prepare data

Prepare your input as an image file or a numpy array, then convert it to a `mediapipe.Image` object. If your input is a video file or live stream from a webcam, you can use an external library such as OpenCV (https://github.com/opencv/opencv) to load your input frames as numpy arrays.

Image Video (#video) Live stream (#live-stream)
(#image)

```
# Load the input image from an image file.
mp_image = mp.Image.create_from_file('/path/to/image')

# Load the input image from a numpy array.
mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=numpy_image)
```

For a code example showing preparation of data for Image Segmenter, see the code example (https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/image_segmentation/python/image_segmentation.ipynb)
.

Run the task

The Image Segmenter uses the `segment`, `segment_for_video` and `segment_async` functions to trigger inferences. For image segmentation, this involves preprocessing input data, running segmentation model and postprocessing the raw model outputs to the segmented masks.

The following code examples show how to execute processing with the task model.

ImageVideo (#video)Live stream (#live-stream)
(#image)

```
# Perform image segmentation on the provided single image.
# The image segmenter must be created with the image mode.
segmented_masks = segmenter.segment(mp_image)
```

Note the following:

- When running in the video mode or the live stream mode, you must also provide the Image Segmenter task the timestamp of the input frame.
- When running in the image or the video model, the Image Segmenter task will block the current thread until it finishes processing the input image or frame.

For a more complete example of running Image Segmenter inferences, see the code example (https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/image_segmentation/python/image_segmentation.ipynb)
.

Handle and display results

The Image Segmenter outputs a list of `Image` data. If `output_type` is `CATEGORY_MASK`, the output is a list containing single segmented mask as an uint8 image. The pixel indicates the recognized category index of the input image. If `output_type` is `CONFIDENCE_MASK`, the output is a vector with size of category number. Each segmented mask is a float image within the range `[0,1]`, representing the confidence score of the pixel belonging to the category.

The following sections show examples of the output data from this task:

Category confidence

The following images show a visualization of the task output for a category confidence mask. The confidence mask output contains float values between `[0, 1]`.

Original image and category confidence mask output. Source image from the Pascal VOC 2012 (https://www.kaggle.com/datasets/huanghanchina/pascal-voc-2012) **dataset.**

Note: In the second, output image, the floating point output values are scaled to values in the `[0, 255]` range and converted to a uint8 data type.

Category value

The following images show a visualization of the task output for a category value mask. The category mask range is `[0, 255]` and each pixel value represents the winning category index of the model output. The winning category index is has the highest score among the categories the model can recognize.

Original image and category mask output. Source image from the Pascal VOC 2012 (https://www.kaggle.com/datasets/huanghanchina/pascal-voc-2012) **dataset.**

Note: In the second, output image, the pixel values are multiplied by 10 to increase contrast and improve the visualization.