Table of Contents

Manual

MobileSAM 2

Getting Started 3

Scripting API

Doji.AI.Segmentation 6

DecoderOutput 7

MobileSAM 9

MobileSAM

Fast Segmentation with Unity Sentis using MobileSAM models.

To get started, please see the Getting Started page. The package also provides a number of samples that can be imported via the Package Manager.

Getting Started

Welcome to the MobileSAM package! This package allows you to perform segmentation tasks in Unity using MobileSAM models. Below is a quick guide on how to get started with this package, including code samples to help you integrate it into your Unity project.

Table of Contents

- Installation
- Basic Setup
- Performing Segmentation
- Full Example

Installation

1. Import the MobileSAM package into your Unity project via OpenUPM.

In `Edit -> Project Settings -> Package Manager` add a new scoped registry:

`Name: Doji` `URL: https://package.openupm.com` `Scope(s): com.doji`

In the Package Manager install `com.doji.mobilesam` either by name or select it in the list under `Package Manager -> My Registries`

Basic Setup

To begin using MobileSAM, you need to create an instance of the `MobileSAM` class. The instance will handle loading the models and initializing the segmentation process.

```
using Doji.AI.Segmentation;

// Initialize MobileSAM
MobileSAM mobileSAMPredictor = new MobileSAM();
```

Performing Segmentation

To perform segmentation on an image, you need to provide the input image as a `Texture` and provide point prompts that guide the segmentation process.

Setting an Image for Segmentation

Before running the segmentation, you must set the image using the SetImage(Texture) method:

```
_mobileSAMPredictor.SetImage(TestImage);
```

This method processes the image and prepares it for segmentation.

Predicting Masks

To predict masks, use the Predict(float[], float[]) method:

```
_mobileSAMPredictor.Predict(pointCoords, pointLabels);
```

- `pointCoords`: An array of point coordinates `[x1, y1, x2, y2, ...]` in pixel values.
- `pointLabels`: An array of labels corresponding to each point (1 for foreground, 0 for background).

Note: Currently, the `box` and `maskInput` parameters are not yet supported.

Retrieving Results

The segmentation result is stored in the `Result` property:

```
RenderTexture resultTexture = _mobileSAMPredictor.Result;
```

You can then use this `RenderTexture` in your Unity application, for example, to display it on a UI element or apply it to a material.

Cleaning Up

Always dispose of the `MobileSAM` instance when you're done using it to free up resources:

```
_mobileSAMPredictor.Dispose();
```

Full Example

Here's a complete example of setting up and using the MobileSAM predictor in a Unity script:

```
using UnityEngine;
using Doji.AI.Segmentation;

public class SegmentationExample : MonoBehaviour
{
    private MobileSAM _mobileSAMPredictor;
    public Texture TestImage;
    public RenderTexture Result;
    private void Start()
    {
        // Initialize the MobileSAM predictor
        _mobileSAMPredictor = new MobileSAM();

        // Set the image for segmentation
        _mobileSAMPredictor.SetImage(TestImage);

        // Perform segmentation
        DoSegmentation();
    }

    private void DoSegmentation()
    {
        if (TestImage == null) {
            Debug.LogError("TestImage is null. Assign a texture to TestImage.");
            return;
        }

        // Example prompt: a single point in the middle of the image
        float[] pointCoords = new float[] { TestImage.width / 2f, TestImage.height / 2f };
        float[] pointLabels = new float[] { 1f };  // 1 for foreground point

        // Perform segmentation
        _mobileSAMPredictor.Predict(pointCoords, pointLabels);

        // Retrieve the result
        Result = _mobileSAMPredictor.Result;

        // Use the result texture (e.g., display on UI, apply to a material, etc.)
    }

    private void OnDestroy()
    {
        // Clean up resources
        _mobileSAMPredictor.Dispose();
    }
}
```

Namespace Doji.AI.Segmentation

Classes

MobileSAM

Predictor using MobileSAM models. Call SetImage(Texture) to specify the image you want to generate masks for. After setting an image you can call Predict(float[], float[]) to get the masks. The results will be stored in the Result RenderTexture.

Structs

DecoderOutput

Struct DecoderOutput

Namespace: Doji.AI.Segmentation

Assembly: Doji.MobileSAM.dll

```
public struct DecoderOutput
```

Inherited Members

ValueType.Equals(object) , ValueType.GetHashCode() , ValueType.ToString() , object.Equals(object, object) , object.GetType() , object.ReferenceEquals(object, object)

Constructors

DecoderOutput(Tensor, Tensor, Tensor)

```
public DecoderOutput(Tensor lowResMasks, Tensor iouPredictions, Tensor masks)
```

Parameters

`lowResMasks` Tensor

`iouPredictions` Tensor

`masks` Tensor

Properties

IoUPredictions

```
public readonly Tensor<float> IoUPredictions { get; }
```

Property Value

Tensor<float>

LowResMasks

```
public readonly Tensor<float> LowResMasks { get; }
```

Property Value

Tensor<float>

Masks

```
public readonly Tensor<float> Masks { get; }
```

Property Value

Tensor<float>

Class MobileSAM

Namespace: Doji.AI.Segmentation

Assembly: Doji.MobileSAM.dll

Predictor using MobileSAM models. Call SetImage(Texture) to specify the image you want to generate masks for. After setting an image you can call Predict(float[], float[]) to get the masks. The results will be stored in the Result RenderTexture.

```
public class MobileSAM : IDisposable
```

Inheritance

object ← MobileSAM

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MobileSAM()

Initializes a new instance of a mask predictor.

```
public MobileSAM()
```

Properties

Backend

Which Unity.Sentis.BackendType to run the model with.

```
public BackendType Backend { get; set; }
```

Property Value

BackendType

Decoder

```
public Model Decoder { get; }
```

Property Value

Model

Encoder

```
public Model Encoder { get; }
```

Property Value

Model

OutputFormat

Which format the predicted masks are returned as.

```
public RenderTextureFormat OutputFormat { get; set; }
```

Property Value

RenderTextureFormat

Result

```
public RenderTexture Result { get; }
```

Property Value

RenderTexture

Methods

Dispose()

```
public void Dispose()
```

Predict(float[], float[])

Predict masks for the given input prompts, using the currently set image.

```
public void Predict(float[] pointCoords, float[] pointLabels)
```

Parameters

`pointCoords` float[]

A float array of point prompts to the model. Each point is given in pixel coordinates.

`pointLabels` float[]

A length N array of labels for the point prompts. 1 indicates a foreground point and 0 indicates a background point.

SetImage(Tensor, (int height, int width))

```
public void SetImage(Tensor transformedImage, (int height, int width) origSize)
```

Parameters

`transformedImage` Tensor

`origSize` (int height, int width)

SetImage(Texture)

Specifies the image to be used for mask generation using the Predict(float[], float[]) method.

```
public void SetImage(Texture image)
```

Parameters

`image` Texture

Remarks

This encodes the input image and stores the calculated image embeddings. The image only needs to be encoded once. Afterwards, you can query as many masks as you want.