Sample Code

Using Core ML for semantic image segmentation

Identify multiple objects in an image by using the DEtection TRansformer image-segmentation model.

Download

iOS 17.0+ | iPadOS 17.0+ | macOS 14.0+ | Xcode 16.0+

Overview

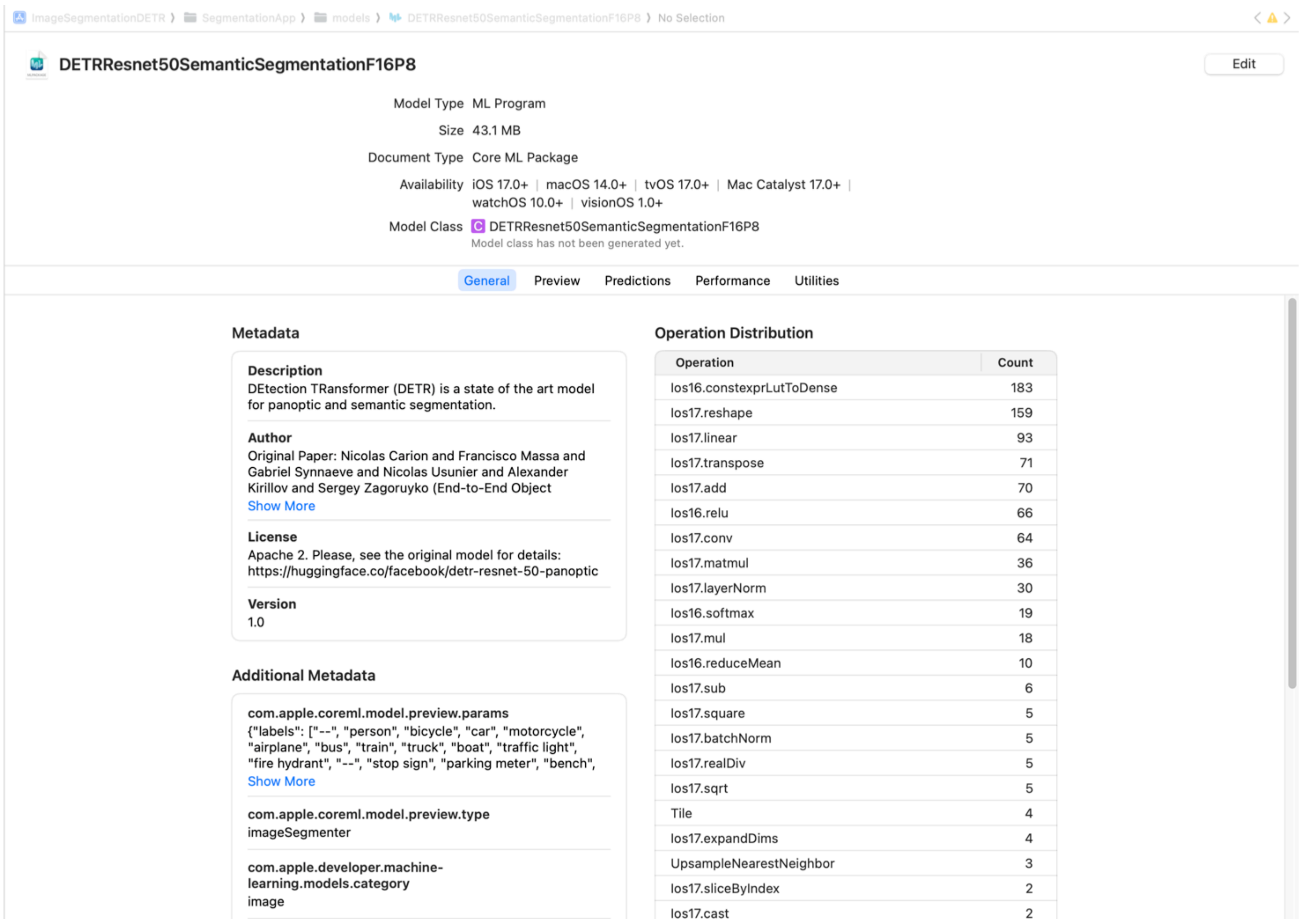
Semantic image-segmentation models identify multiple objects on an input image you provide. For each object, the model output provides the precise locations of each pixel that represents it, so that you can do things like visualize the area of an image that corresponds to the object.

This sample code project shows you how to load the model, read the model's metadata to get label information, and perform offline inference. It also shows how to create a mask — from the image-segment label — and overlay it on the original image.

Select the image-segmentation model

The sample code project includes an 8-bit palletization image-segmentation model — `DETRResnet50SemanticSegmentationF16P8` — in the `models` directory. Replace the model with a different version by dropping the model `.mlpackage` into the `models` folder in Xcode. To download a DETR model using 16-bit, see Core ML Models.

Select the model in the Xcode file navigator to view details in an Xcode model preview area. The following image shows the metadata and operations for the 8-bit model:



ImageSegmentationDETR ❯ SegmentationApp ❯ models ❯ DETRResnet50SemanticSegmentationF16P8 ❯ No Selection

DETRResnet50SemanticSegmentationF16P8

Edit

Model Type	ML Program
Size	43.1 MB
Document Type	Core ML Package
Availability	iOS 17.0+ | macOS 14.0+ | tvOS 17.0+ | Mac Catalyst 17.0+ | watchOS 10.0+ | visionOS 1.0+
Model Class	DETRResnet50SemanticSegmentationF16P8 Model class has not been generated yet.

General   Preview   Predictions   Performance   Utilities

Metadata

Description
DEtection TRansformer (DETR) is a state of the art model for panoptic and semantic segmentation.

Author
Original Paper: Nicolas Carion and Francisco Massa and Gabriel Synnaeve and Nicolas Usunier and Alexander Kirillov and Sergey Zagoruyko (End-to-End Object
Show More

License
Apache 2. Please, see the original model for details: https://huggingface.co/facebook/detr-resnet-50-panoptic

Version
1.0

Additional Metadata

com.apple.coreml.model.preview.params
{"labels": ["--", "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck", "boat", "traffic light", "fire hydrant", "--", "stop sign", "parking meter", "bench",
Show More

com.apple.coreml.model.preview.type
imageSegmenter

com.apple.developer.machine-learning.models.category
image

Operation Distribution

Operation	Count
Ios16.constexprLutToDense	183
Ios17.reshape	159
Ios17.linear	93
Ios17.transpose	71
Ios17.add	70
Ios16.relu	66
Ios17.conv	64
Ios17.matmul	36
Ios17.layerNorm	30
Ios16.softmax	19
Ios17.mul	18
Ios16.reduceMean	10
Ios17.sub	6
Ios17.square	5
Ios17.batchNorm	5
Ios17.realDiv	5
Ios17.sqrt	5
Tile	4
Ios17.expandDims	4
UpsampleNearestNeighbor	3
Ios17.sliceByIndex	2
Ios17.cast	2

To use a model with a different name, but using the same semantics, replace the source code references in `ViewModel` with the model class for `Model` and `ModelOutput`:

```swift
final class ViewModel: ObservableObject {
    typealias Model = DETRResnet50SemanticSegmentationF16P8
    typealias ModelOutput = DETRResnet50SemanticSegmentationF16P8Output
}
```

Load the model

How long it takes to load a model depends on many factors, including the size of the model. The sample app loads the segmentation model asynchronously to avoid blocking the calling thread:

```swift
nonisolated func loadModel() async {
    do {
        let model = try Model()
        let labels = model.model.segmentationLabels
        await didLoadModel(model, labels: labels)
    } catch {
        Task { @MainActor in
            errorMessage = "The model failed to load: \(error.localizedDescription)'
        }
    }
}
```

In the `MainView`, the sample loads the model using the `task` modifier, and displays a progress view to display the load time. The sample uses the `ObservableObject` protocol to observe when the loading is complete:

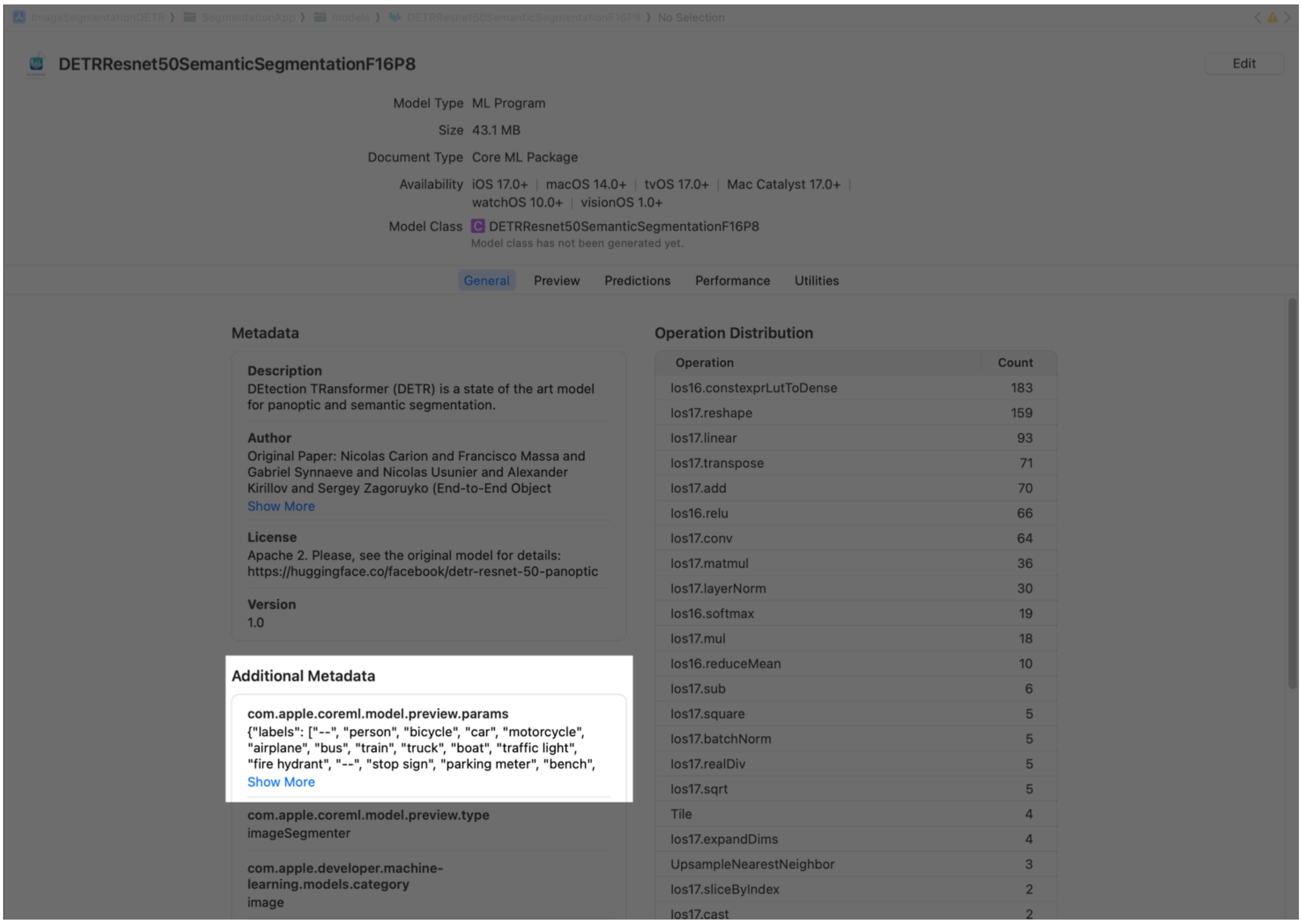
```
var body: some View {
    VStack(spacing: 20) {

        // Other existing view configuration.

        // Display a load message when the sample loads the model.
        if !viewModel.isModelLoaded {
            ProgressView("Loading model...")
        }

        // Display an error message.
        if let message = viewModel.errorMessage, !message.isEmpty {
            Text("Error: \(message)")
        }
    }
    .photosPicker(isPresented: $viewModel.showPhotoPicker, selection: $viewModel.sel
    .task {
        // Load the model asynchronously.
        if loadModel {
            await viewModel.loadModel()
        }
    }
```

Read the image-segmentation model metadata

The `Core ML` framework uses optional metadata to map segment label values into strings an app reads. The metadata is in JSON format, and consists of two optional lists of strings:

- A `label` list that contains the user-readable names for each label
- A `color` list for suggested colors — as hexadecimal RGB codes — an app can use

The following image shows the label metadata in the Xcode model preview area:


ImageSegmentationDETR ❯ SegmentationApp ❯ models ❯ DETRResnet50SemanticSegmentationF16P8 ❯ No Selection

DETRResnet50SemanticSegmentationF16P8

Edit

Model Type ML Program
Size 43.1 MB
Document Type Core ML Package
Availability iOS 17.0+ | macOS 14.0+ | tvOS 17.0+ | Mac Catalyst 17.0+ | watchOS 10.0+ | visionOS 1.0+
Model Class DETRResnet50SemanticSegmentationF16P8
Model class has not been generated yet.

General Preview Predictions Performance Utilities

Metadata

Description
DEtection TRansformer (DETR) is a state of the art model for panoptic and semantic segmentation.

Author
Original Paper: Nicolas Carion and Francisco Massa and Gabriel Synnaeve and Nicolas Usunier and Alexander Kirillov and Sergey Zagoruyko (End-to-End Object
Show More

License
Apache 2. Please, see the original model for details: https://huggingface.co/facebook/detr-resnet-50-panoptic

Version
1.0

Additional Metadata

com.apple.coreml.model.preview.params
{"labels": ["--", "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck", "boat", "traffic light", "fire hydrant", "--", "stop sign", "parking meter", "bench",
Show More

com.apple.coreml.model.preview.type
imageSegmenter

com.apple.developer.machine-learning.models.category
image

Operation Distribution

Operation	Count
Ios16.constexprLutToDense	183
Ios17.reshape	159
Ios17.linear	93
Ios17.transpose	71
Ios17.add	70
Ios16.relu	66
Ios17.conv	64
Ios17.matmul	36
Ios17.layerNorm	30
Ios16.softmax	19
Ios17.mul	18
Ios16.reduceMean	10
Ios17.sub	6
Ios17.square	5
Ios17.batchNorm	5
Ios17.realDiv	5
Ios17.sqrt	5
Tile	4
Ios17.expandDims	4
UpsampleNearestNeighbor	3
Ios17.sliceByIndex	2
Ios17.cast	2

The sample app reads the segmentation labels from the model metadata by calling the `readSegmentationLabels` method in `MLModel`. For more information about the format `Core ML` uses for metadata, see Xcode Model Preview Types:

```
extension MLModel {
    /// The segmentation labels specified in the metadata.
    var segmentationLabels: [String] {
        if let metadata = modelDescription.metadata[.creatorDefinedKey] as? [String:
           let params = metadata["com.apple.coreml.model.preview.params"] as? String
           let data = params.data(using: .utf8),
           let parsed = try? JSONSerialization.jsonObject(with: data) as? [String: A
           let labels = parsed["labels"] as? [String] {
            return labels
        } else {
            return []
        }
    }
}
```

Because the colors from the model metadata are optional, a common practice is to use a generic set of colors that an app defines. A semantic image-segmentation model can generate a large number of labels, and may need to reuse colors or use very similar colors. To assist people with color blindness, avoid using only color to identify different objects. For more information about using inclusive colors, see Inclusive color.

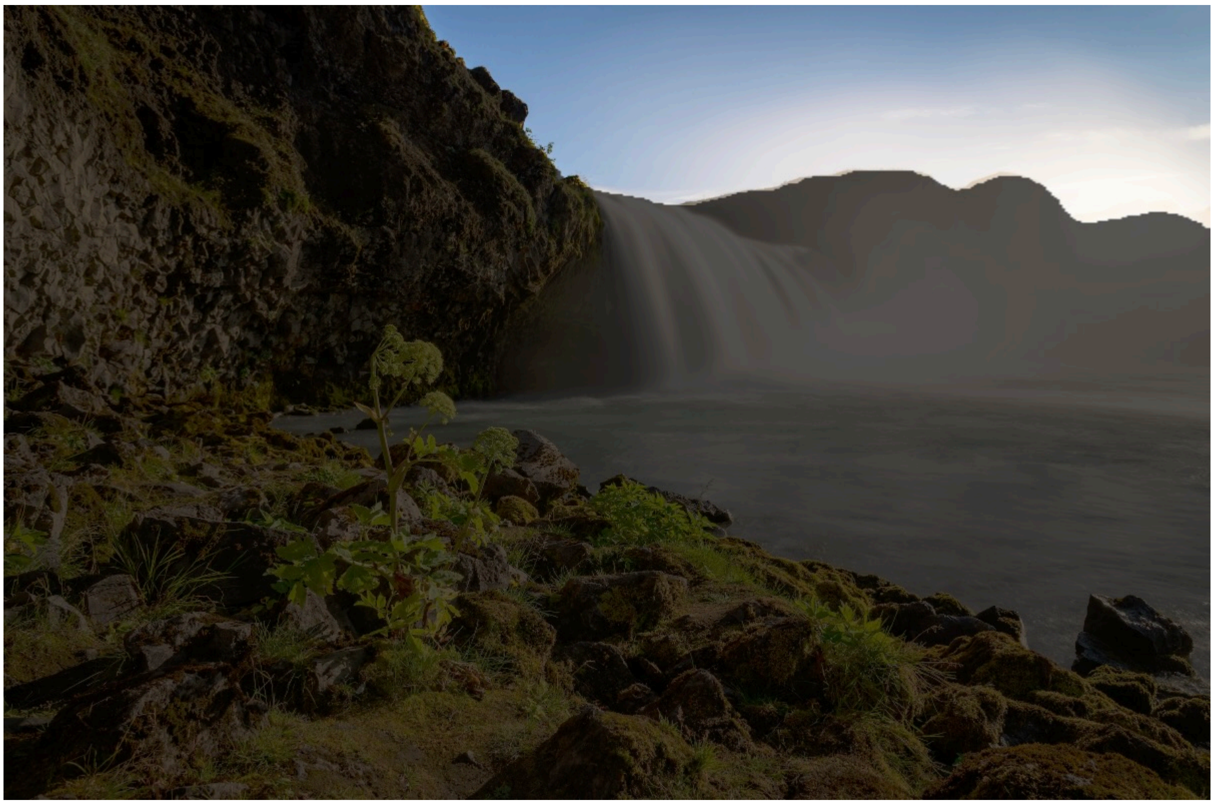
Perform inference

The 8-bit DETR model takes a `CVPixelBuffer` with the type `kCVPixelFormatType_32ARGB` as the input, and generates a `ShapedArray` of `Int32` as output. For both arrays, the model uses fixed dimensions with the size `(448, 448)`. These values are specific to the model, and can be different for another model. To perform inference on the image-segmentation model, the sample app calls the `performInference` method. After inference, the sample app gets the result array from the `semanticPredictionsShapedArray` property:

```
// Resize the input image to the target size.
let resizedImage = await CIImage(cgImage: inputImage.cgImage!).resized(to: targetSiz

// Get the pixel buffer for the image.
let pixelBuffer = context.render(resizedImage, pixelFormat: kCVPixelFormatType_32ARG

// Perform inference.
let result = try model.prediction(image: pixelBuffer)

// Get the result.
let resultArray = result.semanticPredictionsShapedArray
```

Overlay the result

The sample app provides a visualization of model output by overlaying a masked image — representing the selected segment — on top of the original, like the sky in the following image:



Based on the selection, the sample app colors the labeled regions by calling the `renderMask` method in `ViewModel`:

```
func renderMask() throws -> CGImage? {
    guard let resultArray else {
        return nil
    }

    // Convert the results to a mask.
    var bitmap = resultArray.scalars.map { labelIndex in
        let label = self.labelNames[Int(labelIndex)]
        if label == selectedLabel {
            return 0xFFFFFF00 as UInt32
        } else {
            return 0x00000000 as UInt32
        }
    }

    // Convert the mask to an image.
    let width = resultArray.shape[1]
    let height = resultArray.shape[0]
    let image = bitmap.withUnsafeMutableBytes { bytes in
        let context = CGContext(
            data: bytes.baseAddress,
            width: width,
            height: height,
            bitsPerComponent: 8,
            bytesPerRow: 4 * width,
            space: CGColorSpace(name: CGColorSpace.sRGB)!,
            bitmapInfo: CGBitmapInfo.byteOrder32Little.rawValue | CGImageAlphaInfo.r
        )
        return context?.makeImage()
    }

    return image!
```

See Also

Image classification models

{} Classifying Images with Vision and Core ML

Crop and scale photos using the Vision framework and classify them with a Core ML model.

{} Detecting human body poses in an image

Locate people and the stance of their bodies by analyzing an image with a PoseNet model.

{} Understanding a Dice Roll with Vision and Object Detection

Detect dice position and values shown in a camera frame, and determine the end of a roll by leveraging a dice detection model.