Sample Code

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.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 12.0+ | iPad 13.0+

Overview

This sample app uses an object detection model trained with Create ML to recognize the tops of dice and their values when the dice roll onto a flat surface.

After you run the object detection model on camera frames through Vision, the model interprets the result to identify when a roll has ended and what values the dice show.

> **Note**
>
> This sample code project is associated with WWDC 2019 session 228: Creating Great Apps Using Core ML and ARKit.

Configure the sample code project

Before you run the sample code project in Xcode, note the following:

- You must run this sample code project on a physical device that uses iOS 13 or later. The project doesn't work with Simulator.

- The model works best on white dice with black pips. It may perform differently on dice that use other colors.

Add inputs to the request

In Vision, beginning in iOS 13, you can provide inputs other than images to a model by attaching an `MLFeatureProvider` object to your model. This is useful in the case of object detection when you want to specify different thresholds than the defaults.

As shown below, a feature provider can provide values for the `iouThreshold` and `confidenceThreshold` inputs to your object detection model.

To use this threshold provider with your `VNCoreMLModel`, assign it to the `featureProvider` property of your `VNCoreMLModel` as seen in the following example.

Set up a Vision request to handle camera frames

For simplicity, you can use camera frames coming from an `ARSession`.

To run your detector on these frames, first set up a `VNCoreMLRequest` request with your model, as shown in the example below.

Pass camera frames to the object detector to predict dice locations

Pass the frames from the camera to the `VNCoreMLRequest` so it can make predictions using a `VNImageRequestHandler` object. The `VNImageRequestHandler` object handles image resizing and preprocessing as well as post-processing of your model's outputs for every prediction.

To pass camera frames to your model, you first need to find the image orientation that corresponds to your device's physical orientation. If the device's orientation changes, the aspect ratio of the images can also change. Because you need to scale the bounding boxes for the detected objects back to your original image, you need to keep track of its size.

Finally, you invoke the `VNImageRequestHandler` with the image from the camera and information about the current orientation to make a prediction using your object detector.

Now that the app handles providing *input* data to your model, it's time to interpret your model's *output*.

Draw bounding boxes to understand your model's behavior

You can get a better understanding of how well your detector performs by drawing bounding boxes around each object and its text label. The dice detection model detects the tops of dice and labels them according to the number of pips shown on each die's top side.

To draw bounding boxes, see Recognizing Objects in Live Capture.

Determine wwhen a roll has ended

When playing a dice game, users want to know the result of a roll. The app determines that the roll has ended by waiting for the dice's positions and values to stabilize.

You can define the requirements of an ended roll as a comparison between two consecutive camera frames with the following conditions:

- The number of detected dice must be the same.
- For each detected die:
  - The bounding box must have not moved.
  - The identified class must match.

Based on these constraints, you can make a function that tells the app whether a roll has ended based on the current and the previous `VNRecognizedObjectObservation` objects.

Now for every prediction (meaning every new camera frame) you can check whether the roll has ended.

Display the dice values

Once the roll has ended, you can display the information on the screen or trigger some other behavior in the setting of a game.

This sample app shows the list of recognized values on screen, sorted from left-most to right-most `VNRecognizedObjectObservation`. It sorts the values based on where the dice are on the surface according to each observation's bounding box coordinates. The app does this by sorting the observations by their bounding box's `centerX` property in ascending order.

See Also

Image classification models

{} Using Core ML for semantic image segmentation

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

{} 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.