Sample Code

Counting human body action repetitions in a live video feed

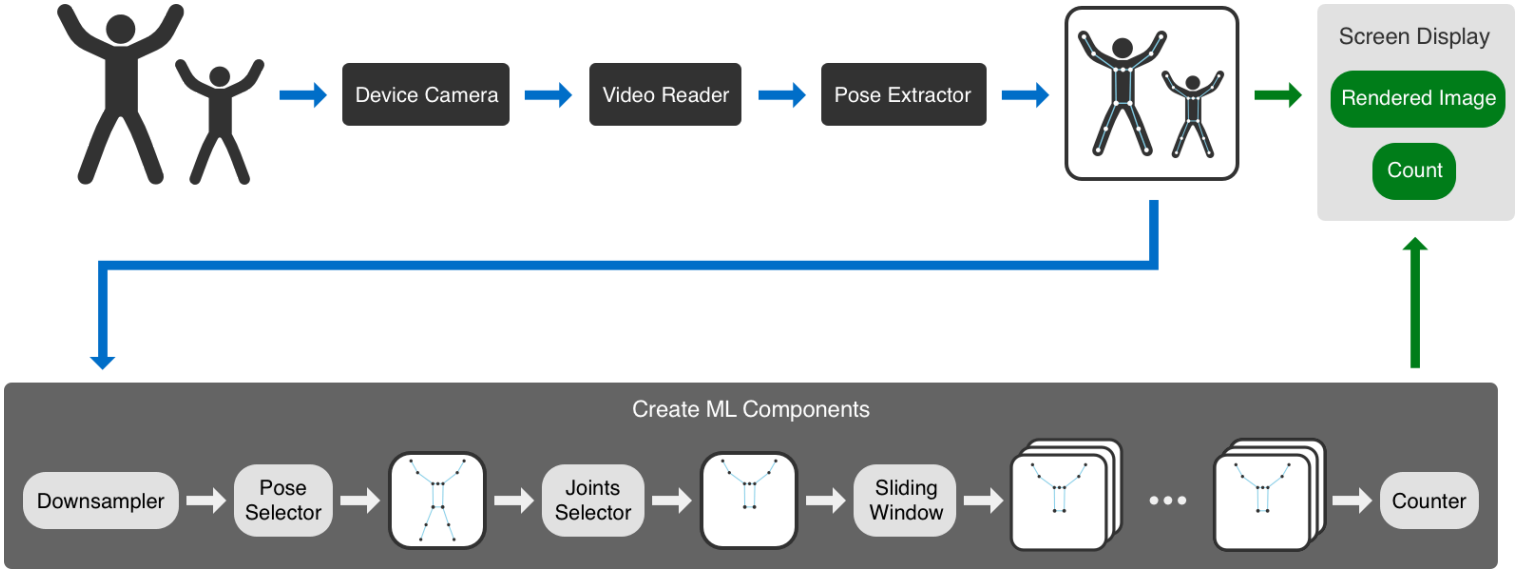
Use Create ML Components to analyze a series of video frames and count a person's repetitive or periodic body movements.

Download

iOS 16.0+ | iPadOS 16.0+ | Xcode 14.0+

Overview

This sample app counts a person's repetitive or periodic body movements (*actions*) by analyzing a series of video frames and making a prediction with a human body action repetition counter. The counter in this sample can count arbitrary body moves that occur at moderate speed, such as jumping jacks, dance spins, and waving arms.



The app continually presents the current action repetition count on top of a live, full-screen video feed from the camera in portrait orientation. When the app detects one or more people in the

frame, it overlays a wireframe body pose on each person. At the same time, the app predicts the action repetition count about the most prominent person across multiple frames, typically whoever is closest to the camera.

The app begins by configuring a camera to generate video frames, then directs the frames through a series of transformers it chains together with Create ML Components. These methods work together to:

1. Read camera frames in real time using `VideoReader`.
2. Analyze each frame to locate any human body poses using `HumanBodyPoseExtractor`, and redirect the pose stream with an AsyncChannel to allow multiple consumers.
3. Optionally, downsample the stream using a `Downsampler` to process the observed actions in different speeds. To improve performance, you can move the downsampler to an earlier stage in the pipeline if you don't need to render poses on every frame.
4. Isolate the prominent pose using `PoseSelector`.
5. Optionally, use `JointsSelector` to select only joints of interest for counting.
6. Aggregate the prominent pose's position data over time using `SlidingWindowTransformer`.
7. Predict action repetitions by sending aggregate data to the `HumanBodyActionCounter`.

> **Note**
>
> This sample code project is associated with WWDC22 session 110332: What's new in Create ML.

Configure the sample code project

This sample code project requires a device with iOS 16 or later, or iPadOS 16 or later. To build this project:

1. Double-click the `CountMyActions.xcodeproj` project to open it in Xcode.
2. In Xcode, from the Project navigator, select the `CountMyActions` project and click the Signing & Capabilities tab.
3. Select your development team from the Add Account pop-up menu.
4. Select your target device from the scheme menu, and choose Product > Run.

Start a live video feed

The app uses `VideoReader` to configure the device's camera and generate an asynchronous video frame sequence. The `VideoReader.CameraConfiguration` specifies the front- or rear-facing camera, and configures its pixel format and resolution. This app supports portrait orientation only. Low lighting and other factors can vary the frame rate, which may affect the counting performance, so ensure the person's full body is visible in bright environments.

```
/// The camera configuration to define the basic camera position, pixel format, and
private var configuration = VideoReader.CameraConfiguration()
```

When the app first launches — or when the user toggles the camera — the video reader configures a camera device, starts the video-processing pipeline, and produces a frame sequence output with `readCamera(configuration:)`.

```
/// Start the video-processing pipeline by displaying the poses in the camera frames
/// starting the action repetition count prediction stream.
func startVideoProcessingPipeline() {

    if let displayCameraTask = displayCameraTask {
        displayCameraTask.cancel()
    }

    displayCameraTask = Task {
        // Display poses on top of each camera frame.
        try await self.displayPoseInCamera()
    }

    if predictionTask == nil {
        predictionTask = Task {
            // Predict the action repetition count.
            try await self.predictCount()
        }
    }
}
```

Analyze each frame for body poses

The `HumanBodyPoseExtractor` is a transformer that can locate any human body poses from an image or a video frame.

```
/// A Create ML Components transformer to extract human body poses from a single ima
private let poseExtractor = HumanBodyPoseExtractor()
```

When the transformation completes, the method creates and returns a `Pose` array that contains one pose for every detected person in the same frame.

```
// Extract poses in every frame.
let poses = try await poseExtractor.applied(to: frame.feature)
```

The `Pose` structure serves the following purposes:

- Calculates the pose's area within a frame (See the "Isolate a body pose" section below.).
- Draws each detected pose as a wireframe of points and lines (See the "Present the poses to the user" section below.).

For more information about the underlying human body pose model, see Detecting Human Body Poses in Images.

Create a pose stream

AsyncChannel sends the extracted poses to a separate asynchronous stream. This allows additional consumers to obtain poses from the upstream asynchronous sequence. `Async Channel` requires the inclusion of the AsyncAlgorithms Swift package.

```
/// An asynchronous channel to divert the pose stream for another consumer.
private let poseStream = AsyncChannel<TemporalFeature<[Pose]>>()
```

Create an action repetition counting pipeline

The `ActionCounter` structure consists of a pipeline of Create ML Components transformers to achieve continuous action repetition counting. It takes a pose stream as input and returns an asynchronous sequence of cumulative counts.

```
/// The counter to count action repetitions from a pose stream.
private let actionCounter = ActionCounter()
```

Downsample a pose stream

The first optional transformer in the pipeline, `Downsampler`, downsamples the incoming pose sequence by an integer factor. This allows the pipeline to process and count much slower actions. For example, without downsampling, the original counter model can handle moderate speed actions, about one repetition per second, such as jumping jacks. A downsampling factor of three can effectively speed up slower actions, such as pushups or a complex dance sequence with about one repetition per 3 seconds, and still allow the model to count the actions.

```
// Use an optional Downsampler transformer to downsample the
// incoming frames (that is, effectively speed up the observed actions).
let pipeline = Downsampler(factor: 1)
```

Isolate a body pose

The next transformer in the pipeline, `PoseSelector`, selects a single pose from the array of poses by using the default strategy, namely, selecting the most prominent person by their maximum bounding box area.

```
// Use a PoseSelector transformer to select one pose to count if
// the system detects multiple poses.
    .appending(PoseSelector(strategy: .maximumBoundingBoxArea))
```

The goal of this strategy is to consistently select the same person's pose from a crowd over time.

Important

Get the most accurate predictions by using whatever strategy best tracks a person from frame to frame.

Select a subset of body joints

The next optional transformer in the pipeline, `JointsSelector`, selects or ignores a specified subset of body joints from the pose.

For example, to count only upper-body movements, the transformer can ignore lower-body joints in the pose, such as knees and ankles, which can eliminate noise by ignoring any leg movements.

```
// Use an optional JointsSelector transformer to specifically ignore
// or select a set of joints in a pose to include in counting.
    .appending(JointsSelector(ignoredJoints: [.nose, .leftEye, .leftEar, .rightEye,
```

Gather a window of poses

The next transformer in the pipeline is a `SlidingWindowTransformer` that receives a pose sequence from its upstream and gathers the frames into an array by providing the following parameters:

- A `stride` that determines the number of frames to count before updating the pose window
- A `length` that determines the window size, namely, how many frames to group together

```
// Use a SlidingWindowTransformer to group frames into windows, and
// prepare them for prediction.
    .appending(SlidingWindowTransformer<Pose>(stride: 5, length: 90))
```

The action repetition counter assumes a fixed length of 90, where the sliding window transformer groups 90 frames together to generate a single prediction count. The stride is adjustable. An example is a stride of 10 frames, indicating the count updates every 10 frames, which is about 0.3 seconds if the frame rate is 30 frames per second. When the stride is smaller than the length, the windows overlap.

Predict the person's action repetition count

The next transformer in the pipeline, `HumanBodyActionCounter`, takes a stream of grouped pose windows as input and produces a `HumanBodyActionCounter.CumulativeSum Sequence` where each result is a cumulative count of the actions in the sequence. Live counting occurs by iterating each item in the resulted sequence.

```
// Use a HumanBodyActionCounter transformer to count actions from
// each window and produce cumulative counts for the input stream.
    .appending(HumanBodyActionCounter())
```

Present the count to the user

The final count appears as a SwiftUI label on the screen using the `OverlayView` structure on the main thread.

Present the poses to the user

The app visualizes the result of each detected human body pose by drawing the poses on top of the frame that `HumanBodyPoseExtractor` finds them in. Each time the `poseExtractor` creates an array of `Pose` instances, the `PosesView` iterates each detected pose and draws it by calling its `drawWireframe(to:applying:)` method, which draws the pose as a wireframe of connection lines and joint circles.

```
// Draw all the poses Vision finds in the frame.
for pose in poses {
    // Draw each pose as a wireframe at the scale of the image.
    pose.drawWireframe(to: context, applying: pointTransform)
}
```

The `ViewModel` presents the image and poses onscreen by calling `display(image:, poses:)` method.

See Also

Pose components

`struct Pose`

A pose that contains joint keypoints from a person, a hand, or a combination.

`struct JointKey`

A key that uniquely identifies a joint.

`struct JointPoint`

A joint in a pose that contains a location and scoring information.

`struct PoseSelector`

A transformer that selects one pose from an array of poses.

`enum PoseSelectionStrategy`

Pose selection strategy.

`struct JointsSelector`

Joints selector from a pose.

`struct HumanBodyPoseExtractor`

The human body pose image feature extractor.

`struct HumanHandPoseExtractor`

The human hand pose image feature extractor.

`struct HumanBodyActionCounter`

A human body action repetition counting transformer that takes window of human body poses and produces cumulative human body action repetition counts.

`struct HumanBodyActionPeriodPredictor`

A human body action period predictor transformer that takes window of poses and produces a window of predictions.