Sample Code

Generating high-quality thumbnails from videos

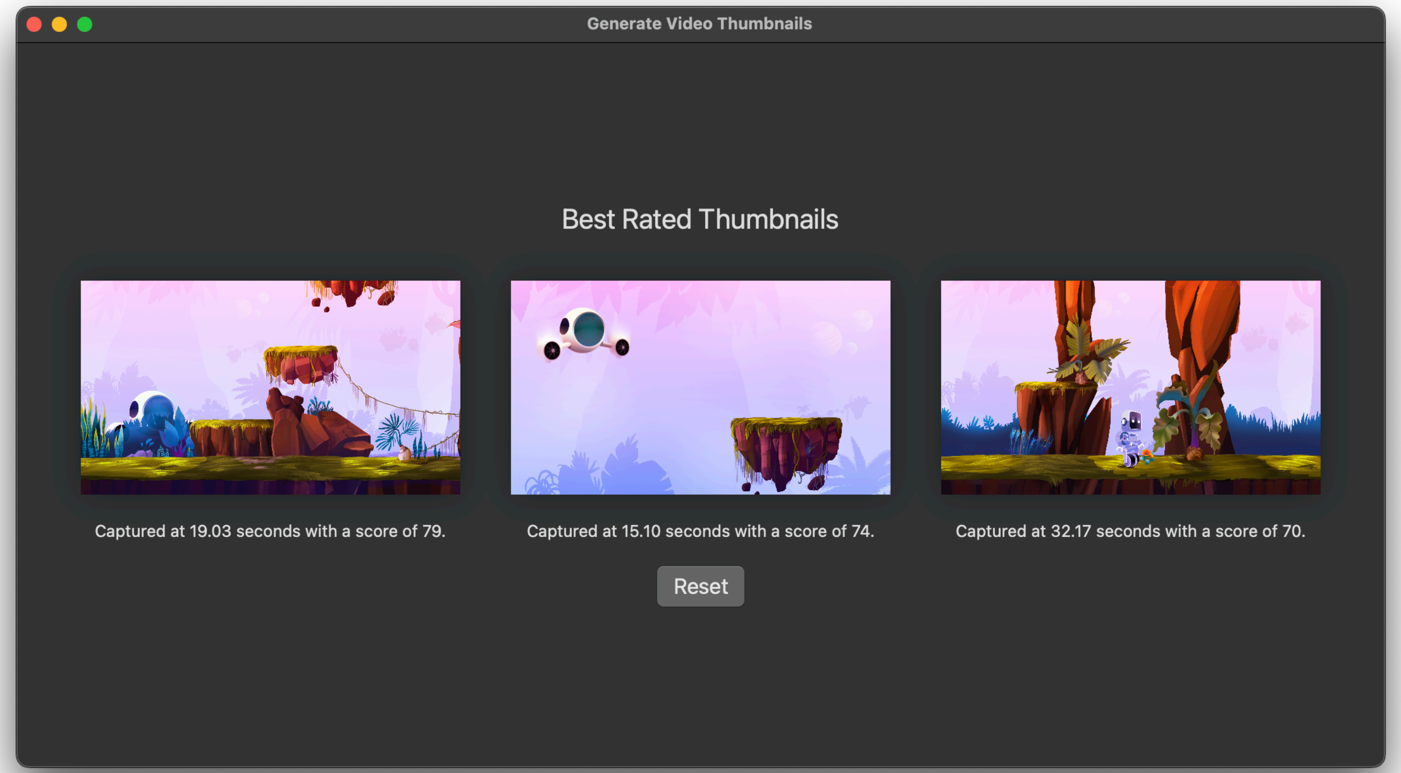
Identify the most visually pleasing frames in a video by using the image-aesthetics scores request.

Download

iOS 18.0+ | iPadOS 18.0+ | macOS 15.0+ | visionOS 2.0+ | Xcode 16.0+

Overview

Using the Vision framework, you can process images and videos frame by frame, enabling image-analysis requests through `VisionRequest`. The sample demonstrates how to process video input to identify the three frames with the highest aesthetic scores. By leveraging this capability, you can automatically select the best frames for creating promotional materials or for highlighting short films. The following image shows a preview of the sample:


Generate Video Thumbnails
Best Rated Thumbnails
Captured at 19.03 seconds with a score of 79.
Captured at 15.10 seconds with a score of 74.
Captured at 32.17 seconds with a score of 70.
Reset

The sample uses a `VideoProcessor` to convert the video into a stream of frames, which `CalculateImageAestheticsScoresRequest` analyzes to rate each frame with an overall aesthetic score. To avoid selecting similar thumbnails, the `GenerateImageFeaturePrintRequest` compares the image similarity. Finally, AVFoundation extracts the images from the selected frames and presents them as thumbnails.

Set up the representation of frames and thumbnails

The `Frame` structure contains the information about a specific frame in the video that the sample uses to determine the best frame to use as a thumbnail. It accepts a `CMTime` to represent the timestamp of the frame, a `Float` value for the overall score of the frame, and a `FeaturePrintObservation` to enable comparison to other frames:

```swift
struct Frame {
    /// The timestamp of the frame.
    let time: CMTime

    /// The score of the frame.
    let score: Float

    /// The feature-print observation of the frame.
    let observation: FeaturePrintObservation
}
```

To present the results of each frame, the sample creates a `Thumbnail` class that conforms to the `Identifiable` protocal. This ensures that the class has a unique identity for `ForEach` to display the results. This class accepts a `CGImage` to store the image of the frame, and a `Frame` to establish a connection between the frame and the thumbnail:

```swift
class Thumbnail: Identifiable {
    /// The image that captures from the video frame.
    let image: CGImage

    /// The frame that the thumbnail represents.
    let frame: Frame

    // ...
}
```

Process the video

The sample processes videos frame by frame using the `VideoProcessor`. The sample first initializes the video processor with the video URL, then create the instances of the requests the sample uses to process the video:

```swift
func processVideo(for videoURL: URL, progression: Binding<Float>) async -> [Thumbnai
    /// The instance of the `VideoProcessor` with the local path to the video file.
    let videoProcessor = VideoProcessor(videoURL)

    /// The request to calculate the aesthetics score for each frame.
    let aestheticsScoresRequest = CalculateImageAestheticsScoresRequest()

    /// The request to generate feature prints from an image.
    let imageFeaturePrintRequest = GenerateImageFeaturePrintRequest()

    /// The array to store information for the frames with the highest scores.
    var topFrames: [Frame] = []

    // ...
}
```

The `CalculateImageAestheticsScoresRequest` calculates the aesthetic scores for each frame. To ensure that the thumbnail results represent different scenes rather than variations of the same frame, `GenerateImageFeaturePrintRequest` computes the similarity between the frames.

The sample calculates a time interval for the video processor to process approximately 100 frames, adds the `aestheticsScoresRequest` and `imageFeaturePrintRequest` to the video processor, then starts the video-analysis process. To store the timestamp and the results from the video-processor stream, the sample creates two dictionaries: `aestheticsResults` and `featurePrintResults`:

```
do {
    /// The time interval for the video-processing cadence.
    let interval = CMTime(
        seconds: totalDuration / framesToEval,
        preferredTimescale: timeScale
    )

    /// The video-processing cadence to process only 100 frames.
    let cadence = VideoProcessor.Cadence.timeInterval(interval)

    /// The stream that adds the aesthetics scores request to the video processor.
    let aestheticsScoreStream = try await videoProcessor.addRequest(aestheticsScores

    /// The stream that adds the image feature-print request to the video processor.
    let featurePrintStream = try await videoProcessor.addRequest(imageFeaturePrintRe

    // Start to analyze the video.
    videoProcessor.startAnalysis()

    /// The dictionary to store the timestamp and the aesthetics score.
    var aestheticsResults: [CMTime: Float] = [:]

    /// The dictionary to store the timestamp and the feature-print observation.
    var featurePrintResults: [CMTime: FeaturePrintObservation] = [:]
    // ...

    // Solve for the top-rated frames.
    topFrames = await calculateTopFrames(aestheticsResults: aestheticsResults, featu
}
```

When the sample receives the results for both requests, call `calculateTopFrames(aestheticsResults:featurePrintResults:)` to solve for the top-rated frames.

Solve for the top-rated frames

The function uses `aestheticsResults` and `featurePrintResults` to identify three frames that have the highest aesthetic scores and are different from each other:

```
for (time, score) in aestheticsResults {
    /// The `FeaturePrintObservation` for the timestamp.
    guard let featurePrint = featurePrintResults[time] else { continue }

    /// The new frame at that timestamp.
    let newFrame = Frame(time: time, score: score, observation: featurePrint)

    /// The variable that tracks whether to add the image based on image similarity.
    var isSimilar = false

    // Iterate through the current top-rated frames to check whether any of them
    // are similar to the new frame and find the insertion index.
    for (index, frame) in topFrames.enumerated() {
        if let distance = try? featurePrint.distance(to: frame.observation), distanc
            // Replace the frame if the new frame has a higher score.
            if newFrame.score > frame.score {
                topFrames[index] = newFrame
            }
            isSimilar = true
            break
        }

        // ...
    }
}
```

For each result, the sample creates a new frame based on the timestamp and attaches the score and `FeaturePrintObservation`. It checks for similar frames in `topFrames` with the `distance(to:)` function to compare the observations. If there is a match, the sample keeps the frame with the higher score and exits the loop.

See Also

Image aesthetics analysis

`struct CalculateImageAestheticsScoresRequest`

A request that analyzes an image for aesthetically pleasing attributes.