Machine Learning

Ray

Use CoreML, CreateML, turi

Input:
- cameras
  - photos or videos
  - objects, faces, landmarks
  - handwriting, printed text
  - track motion and poses
  - recognize gestures
  - understand emotional cues in video
- text
  - meaning
- speech
  - convert to text
  - translation
  - siri instructions
- activity
  - gyroscopes
  - altimeter
  - magnetometer
  - accelerometer
  - gps

ML Model
- *learned* algorithm for specific task, plus data needed to run it

Recognizing faces in photos:
- *domain* is digital photos in humans
- *model* is everything you need to make sense of the photos
Create model
• choose algorithm
• train it

After training, model contains knowledge about the problem extracted from the training examples. Then it can answer questions, including about things it has not been told:
• this is called inference
• it “generalizes well” if it can make correct predictions about things it was not trained on.

Types:
• supervised - needs labeled data (“this picture is Ann”)
  ◦ classification
    • spam
    • names of people in pictures
• unsupervised

How applicable are the pre-trained models?
• maybe pretty good - convert to CoreML, pop it in, and go
• maybe you want it to make decisions slightly differently
• with tagging people in photos the model won’t know anything about your friends a priori

Core ML default models comes w/:
• thousands of different features
• one thousand different classes of objects
Training from scratch is obviously difficult, and expensive.

So...take existing pre-trained model, modify for your dataset: transfer learning. Much faster, dataset much smaller, small amount of training for feature specific to your use case.

Apple ML has two tools to help w/ transfer learning:
• CreateML
• turi
Binary classification examples:
- cat/dog
- has covid/doesn’t have covid
- spam

CoreML packages up structure of model, together with parameters, or weights, it has learned.

Starting point is that we have a pre-trained model. Add to project and look at it:
- neural nets (implies deep learning)
- size
- inputs/outputs
  - 227x227 input
  - two value/weight pairs

Vision

Can set up vision pipelines. For example:
- use vision to recognize faces
- hand off each face to ML for happy/sad analysis

Can also automatically:
- crop and re-size
- rotate for orientation
- adjust color palette to match device

Classification requests

- typically use the same request for all your images, so build once (and lazily)

```swift
lazy var classificationRequest: VNCoreMLRequest = {
    let visionModel = try! VNCoreMLModel(for: HealthySnacks().model)
    let request = VNCoreMLRequest(model: visionModel)
    request.imageCropAndScaleOption = .centerCrop
```
There are actually three crop/scale options:
- centerCrop
- scaleFill (keeps all pixels, might distort)
- scaleFit (black bars if not square)
Want to use the same option for your requests as for the original training.

**Classification**

```swift
func classify(image: UIImage) {
    DispatchQueue.global(qos: .userInitiated).async {
        let ciImage = CIImage(image: image)!
        let orientation = CGImagePropertyOrientation(image.imageOrientation)
        let handler = VNImageRequestHandler(ciImage: ciImage, orientation: orientation)
        try! handler.perform([self.classificationRequest])
    }
}
```

Vision can use either UIImage or CIImage, but some of the translations and mutations work only on CIImage, so good to convert over.

iOS camera always stores images in landscape
- this is the sensor orientation
- iOS does track true orientation of camera (exif imagedata)
- coreML does not accept image orientation as input
  - though horizontal flips expected so cat looking left same as cat looking right

**Queues: When and Where**

ML is time-consuming, so processing done off main queue.
- .userInitiated
results to back on main queue

```swift
func processObservations(for request: VNRequest) {
    DispatchQueue.main.async {
        let result = request.results!.first as! VNClassificationObservation
        let formatter = NumberFormatter()
        formatter.maximumFractionDigits = 1
        let confP = formatter.string(from: result.confidence*100 as NSNumber)
        self.resultsLabel.text = "\(result.identifier) \(confP)%"
        self.showResultsView()
    }
}
```

Overall pattern common:
- single request, re-purposed for each image
- different closures on lower QoS queue for each image
- show results back on main queue

Results are going to be observations for each class:
- for this example: healthy, unhealthy
- sorted in descending order
  - first is our decision/guess

We’re going to call `processObservations()` from the completion handler for the requests:

```swift
lazy var classificationRequest: VNCoreMLRequest = {
    let visionModel = try! VNCoreMLModel(for: HealthySnacks().model)
    let request = VNCoreMLRequest(model: visionModel) {
        unowned self
    }
    request, _ in
```
```swift
self.processObservations(for: request)
```n
```swift
request.imageCropAndScaleOption = .centerCrop
return request
}
```n
Second parameter to completion handler is an error value, ignored for tutorial.

Try It

Really off-the-wall objects

- give confident numbers very near 50% for our binary classifier.
- set a threshold, say 75%, and respond with an error message if the confidence in the first choice is not higher than the threshold. See:
  - precision-recall curves
  - receiver operator characteristic (ROC) curves

Only works well on objects similar to training objects
- can fail badly (high confidence, but wrong) on anything else
- can add more training data, more categories

Confidence Thresholds

```swift
func processObservations(for request: VNRequest) {
    DispatchQueue.main.async {
        let result = request.results!.first as! VNClassificationObservation
        self.resultsLabel.text = result.confidence > 0.8 ? {
            let formatter = NumberFormatter()
            formatter.maximumFractionDigits = 1
            let confP = formatter.string(from: result.confiden
case*100 as NSNumber)
```
return "\(\text{result.identifier} ) \text{(confP)}\%

}() : "Not Sure!"

self.showResultsView()
}
}