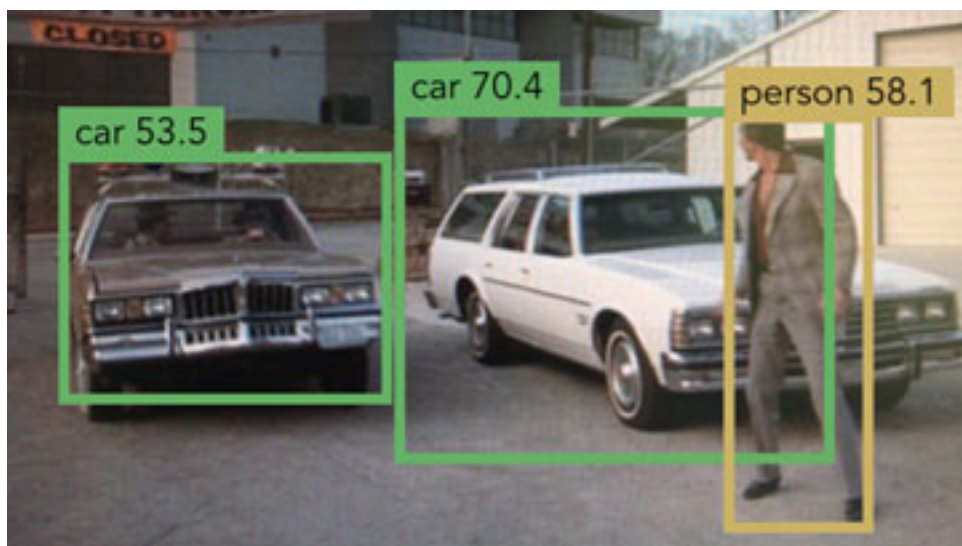

YOLO with Core ML and MPSNNGraph

This is the source code for my blog post [YOLO: Core ML versus MPSNNGraph](#).

YOLO is an object detection network. It can detect multiple objects in an image and puts bounding boxes around these objects. Read my other blog post about YOLO to learn more about how it works.



Previously, I implemented YOLO in Metal using the Forge library. Since then Apple released Core ML and MPSNNGraph as part of the iOS 11 beta. So I figured, why not try to get YOLO running on these two other technology stacks too?

In this repo you'll find:

- **TinyYOLO-CoreML:** A demo app that runs the Tiny YOLO neural network on Core ML.
- **TinyYOLO-NNGraph:** The same demo app but this time it uses the lower-level graph API from Metal Performance Shaders.
- **Convert:** The scripts needed to convert the original DarkNet YOLO model to Core ML and MPS format.

To run the app, just open the **xcodeproj** file in Xcode 9 or later, and run it on a device with iOS 11 or better installed.

The reported “elapsed” time is how long it takes the YOLO neural net to process a single image. The FPS is the actual throughput achieved by the app.

NOTE: Running these kinds of neural networks eats up a lot of battery power. To measure the maximum speed of the model, the `setUpCamera()` method in `ViewController.swift` configures the camera to run at 240 FPS, if available. In a real app, you'd use at most 30 FPS and possibly

limit the number of times per second it runs the neural net to 15 or less (i.e. only process every other frame).

Tip: Also check out this repo for YOLO v3. It works the same as this repo, but uses the full version of YOLO v3!

iOS 12 and VNRecognizedObjectObservation

The code in my blog post and this repo shows how take the `MLMultiArray` output from TinyYOLO and interpret it in your app. That was the only way to do it with iOS 11, but as of iOS 12 there is an easier solution.

The Vision framework in iOS 12 directly supports YOLO-like models. The big advantage is that these do the bounding box decoding and non-maximum suppression (NMS) inside the Core ML model. All you need to do is pass in the image and Vision will give you the results as one or more `VNRecognizedObjectObservation` objects. No more messing around with `MLMultiArray`s.

It's also really easy to train such models using Turi Create. It combines TinyYOLO v2 and the new `NonMaximumSuppression` model type into a so-called pipeline model.

The good news is that this new Vision API also supports other object detection models!

I added a chapter to my book Core ML Survival Guide that shows exactly how this works. In the book you'll see how to add this same functionality to **MobileNetV2 + SSDLite**, so that you get `VNRecognizedObjectObservation` predictions for that model too. The book has lots of other great tips on using Core ML, so check it out! :smile:

If you're not ready to go all-in on iOS 12 yet, then read on...

Converting the models

NOTE: You don't need to convert the models yourself. Everything you need to run the demo apps is included in the Xcode projects already.

If you're interested in how the conversion was done, there are three conversion scripts:

YAD2K

The original network is in Darknet format. I used YAD2K to convert this to Keras. Since `coremltools` currently requires Keras 1.2.2, the included YAD2K source code is actually a modified version that runs on Keras 1.2.2 instead of 2.0.

First, set up a virtualenv with Python 3:

```
1 virtualenv -p /usr/local/bin/python3 yad2kenv
2 source yad2kenv/bin/activate
3 pip3 install tensorflow
4 pip3 install keras==1.2.2
5 pip3 install h5py
6 pip3 install pydot-ng
7 pip3 install pillow
8 brew install graphviz
```

Run the `yad2k.py` script to convert the Darknet model to Keras:

```
1 cd Convert/yad2k
2 python3 yad2k.py -p ../tiny-yolo-voc.cfg ../tiny-yolo-voc.weights
  model_data/tiny-yolo-voc.h5
```

To test the model actually works:

```
1 python3 test_yolo.py model_data/tiny-yolo-voc.h5 -a model_data/tiny-
  yolo-voc_anchors.txt -c model_data/pascal_classes.txt
```

This places some images with the computed bounding boxes in the `yad2k/images/out` folder.

coreml.py

The **coreml.py** script takes the `tiny-yolo-voc.h5` model created by YAD2K and converts it to `TinyYOLO.mlmodel`. Note: this script requires Python 2.7 from `/usr/bin/python` (i.e. the one that comes with macOS).

To set up the virtual environment:

```
1 virtualenv -p /usr/bin/python2.7 coreml
2 source coreml/bin/activate
3 pip install tensorflow
4 pip install keras==1.2.2
5 pip install h5py
6 pip install coremltools
```

Run the `coreml.py` script to do the conversion (the paths to the model file and the output folder are hardcoded in the script):

```
1 python coreml.py
```

nngraph.py

The **nngraph.py** script takes the [tiny-yolo-voc.h5](#) model created by YAD2K and converts it to weights files used by [MPSNNGraph](#). Requires Python 3 and Keras 1.2.2.