Real-time Object Tracking on Resource-constrained Device: MobileNet

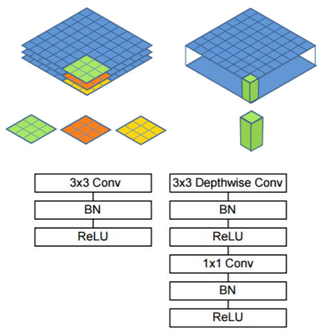
Yundong Zhang Pan Hu Haomin Peng
{yundong, panhu, haomin}@stanford.edu

Motivation

In this project, we aim at deploying a real-time object detection system that operates at high FPS on resource-constrained device such as Raspberry Pi and mobile phones. Object detection can be applied in many scenarios, among which traffic surveillance is particularly interesting to us due to its popularity in daily life. Although many systems have proved their success since the era of machine learning and neural network, most of the evaluations are done with high-end CPU or GPU. Nevertheless, real-world applications such as surveillance pose strict constraints on the resources of the device: low-power, small form factor, relatively good accuracy and fast speed, making it hard to find a good trade-off when designing the system. We present an object detection pipeline which is capable of working smoothly in the situation of traffic surveillance on Raspberry Pi 3 with only 1GB RAM and 1.2GHz ARM CPU that costs merely $35.99.

MobileNet Model

The backbone of our system is MobileNet, a novel deep NN model proposed by Google, designed specifically for mobile vision applications. The main thing that makes it stand out is the use of depth-wise separable (DW-S) convolution.



Convolution layer	Parameter Size	Computation Cost
Standard	KxKxDxD'	KxKxFxFxDx D'
DW-S	KxKxD+DxD'	KxKxFxFxD+ DxD'xFxF

Left: Depthwise Convolution layer structure
Right: Compartion of normal convolutional layer and deepwise convolution layer

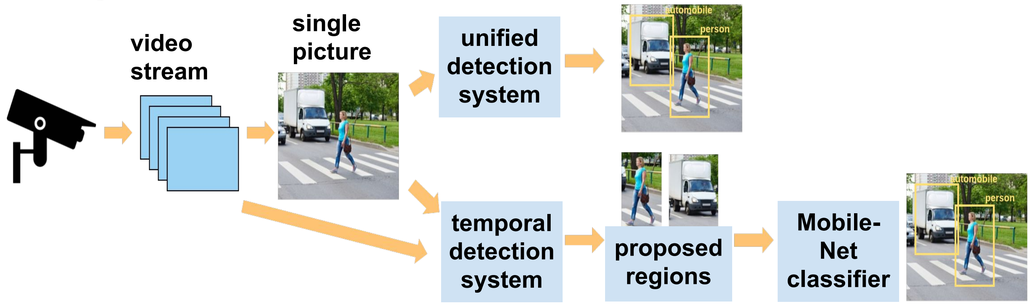
Data Preprocessing

- Our image data was supplied by VOC2012 detection dataset of 20 classes and of various picture size, with training size of 5717 pictures and validation set of 5823 pictures. To guarantee that each picture only contains one region of interest, for pictures with multiple objects, we generated a series of pictures with one object by covering the other object with mean pixel.
- This step generated a 13609 pictures training set and 13841 pictures validation set. After that, we resized each picture to 224x224 pixel.

Detection Pipeline

Initialization process:

- Config camera as video stream
- Take one frame from the stream as reference frame (will update it with momentum as system runs)
- Initialized dependencies: OpenCV, average frame, TensorFlow weights.



Region proposal:

- Get frame difference by subtracting current frame with reference
- Pass diff. frame through Gaussian filter to smooth out noise
- Pass diff. frame through Erode filter to remove smaller area of noise
- Pass diff. frame through Dilated filter to connect close-separated region
- Find contour on difference frame
- Construct rectangle bounding box based on contour

Classifier

- Crop frame according to bounding box
- Resize cropped image to 224*224 Or
- Resize cropped image to 32*32 Or
- If image is larger than 224*224, resize the image but keep aspect ratio.If image is smaller than 224*244, place image in center of the frame. Fill rest of image with average value
- Feed image into classifier
- Draw bounding box and classification label over the frame.

Evaluation

	Speed (fps)	Accuracy(mAP)	Model Size
full-Yolo	OOM	0.6847	269.9Mb
8-bit quantized full-yolo	0.153	0.6133	64.4Mb
tiny-yolo	0.487	0.5514	60.5Mb
Ours	2.566	Classification acc: 67.9%	4.4Mb

We evaluate several systems on Raspberry Pi 3, which has four built-in ARM Cortex-A53 processing cores. All the following models are trained from Pascal-VOC object detection datasets. We also built a prototype system for demonstration in this poster session, feel free to play with it.

Eight-bit Quantization

- For each layer, compute the maximum, minimum and step: (max-min)/255
- Quantize weights into its eight-bit version
- Doing computation in int16;
- dequantize to float8

Future works

- Benchmark our system our credentialed datasets;
- Extend MobileNet to Detection framework(e.g. SSD, faster-rcnn)