You Only Look Once

Introduction

You only look once (YOLO) is a system for detecting objects on the Pascal VOC 2012 dataset. It can detect the 20 Pascal object classes:

person

bird, cat, cow, dog, horse, sheep

aeroplane, bicycle, boat, bus, car, motorbike, train

bottle, chair, dining table, potted plant, sofa, tv/monitor

Detection Using A Pre-Trained Model

We have to download the pre-trained weight file here.
Now run the Darknet yolo command in testing mode:

```
./darknet yolo test cfg/yolo.cfg <path>/yolo.weights <image>
```

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. You can find the source on GitHub.

Darknet prints out the objects it detected, its confidence, and how long it took to find them. Since we are using Darknet on the CPU it takes around 6-12 seconds per image. If we use the GPU version it would be much faster.

YOLO Model Comparison

yolo.cfg is based on the extraction network. It processes images at 45 fps, here are weight files for yolo.cfg trained on 2007 train/val+ 2012 train/val, and trained on all 2007 and 2012 data.	yolo-small.cfg has smaller fully connected layers so it uses far less memory. It processes images at 50 fps, here are weight files for yolo-small.cfg trained on 2007 train/val+ 2012 train/val.	yolo-tiny.cfg is much smaller and based on the Darknet reference network. It processes images at 155 fps, here are weight files for yolo-tiny.cfg trained on 2007 train/val+ 2012 train/val.

Changing The Detection Threshold

By default, YOLO only displays objects detected with a confidence of .2 or higher. You can change this by passing the -thresh <val> flag to the yolo command. For example, to display all detection you can set the threshold to 0:

```
./darknet yolo test cfg/yolo.cfg yolo.weights data/dog.jpg -thresh 0
```

Real-Time Detection

To run this demo you will need to compile Darknet with CUDA and OpenCV. You will also need to pick a YOLO config file and have the appropriate weights file. Then run the command:

```
./darknet yolo demo cfg/yolo.cfg yolo.weights
```

YOLO will display the current FPS and predicted classes as well as the image with bounding boxes drawn on top of it.

Training YOLO

You can train YOLO from scratch if you want to play with different training regimes, hyper-parameters, or datasets.

Get The Pascal VOC Data

To train YOLO you will need all of the VOC data from 2007 to 2012. You can find links to the data here.

Generate Labels for VOC

Now we need to generate the label files that Darknet uses. Darknet wants a `.txt` file for each image with a line for each ground truth object in the image that looks like:

```
<object-class> <x> <y> <width> <height>
```

Where `x`, `y`, `width`, and `height` are relative to the image's width and height. To generate these file we will run the `voc_label.py` script in Darknet's `scripts/` directory.

Darknet needs one text file with all of the images you want to train on. In this example, let's train with everything except the validation set from 2012 so that we can test our model. Run:

```
cat 2007_* 2012_train.txt > train.txt
```

Now we have all the 2007 images and the 2012 train set in one big list. That's all we have to do for data setup!

Point Darknet to Pascal Data

Now go to your Darknet directory. We will have to change the `train` subroutine of `yolo` to point it to your copy of the VOC data. Edit `src/yolo.c`, lines 54 and 55:

```
57    char *train_images = "/home/pjreddie/data/voc/test/train.txt";
58    char *backup_directory = "/home/pjreddie/backup/";
```

`train_images` should point to the `train.txt` file you just generated and `backup_directory`should point to a directory where you want to store backup weights files during training. Once you have edited the lines, re-`make` Darknet.

Download Pretrained Convolutional Weights

For training we use convolutional weights that are pre-trained on Imagenet. We use weights from the Extraction model. You can just download the weights for the convolutional layers here (54 MB).

If you want to generate the pre-trained weights yourself, download the pretrained Extraction model and run the following command:

```
./darknet partial cfg/extraction.cfg extraction.weights extraction.conv.weights 25
```

But if you just download the weights file it's way easier.

Train!!

You are finally ready to start training. Run:

```
./darknet yolo train cfg/yolo.cfg extraction.conv.weights
```

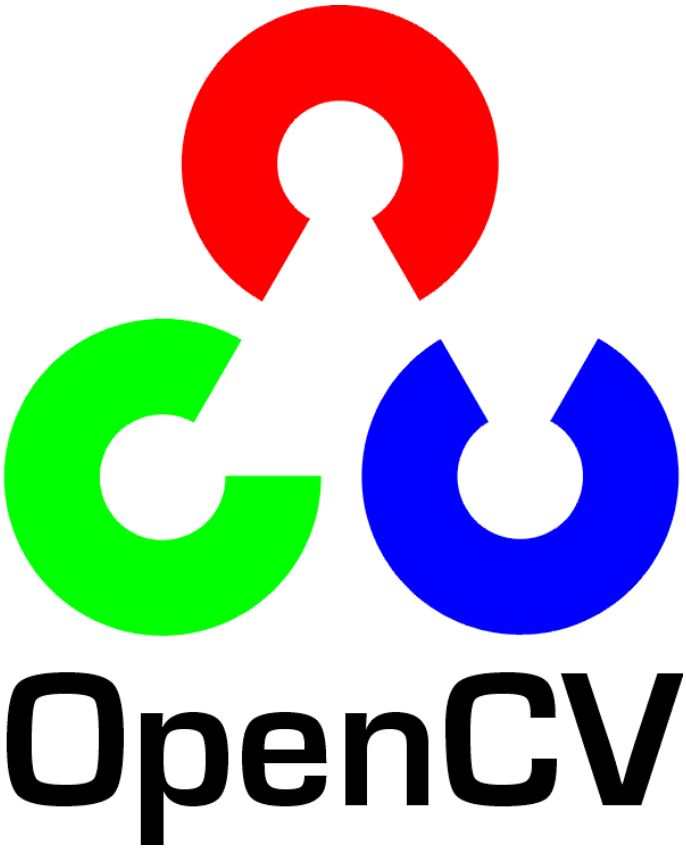
It should start spitting out numbers and stuff.

If you want it to go faster and spit out fewer numbers you should stop training and change the config file a little. Modify `cfg/yolo.cfg` so that on line 3 it says `subdivisions=2` or 4 or something that divides 64 evenly. Then restart training as above.

Training Checkpoints

After every 128,000 images Darknet will save a training checkpoint to the directory you specified in `src/yolo.c`. These will be titled something like `yolo_12000.weights`. You can use them to restart training instead of starting from scratch.

After 40,000 iterations (batches) Darknet will save the final model weights as yolo_final.weights. Then you are done!



Introduction

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Morphological Operations

- A set of operations that process images based on shapes. Morphological operations apply a *structuring element* to an input image and generate an output image.
- The most basic morphological operations are two: Erosion and Dilation.

Erosion

compute a local minimum over the area of the kernel.

The erosion makes the object in white smaller.

It used to remove vertical lines

Canny Edge Detector

Canny algorithm aims to satisfy three main criteria:

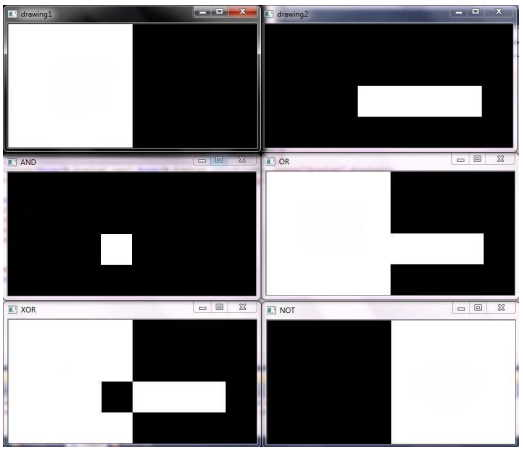
- **Low error rate:** Meaning a good detection of only existent edges.
- **Good localization:** The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response:** Only one detector response per edge.

Used in edge detection.

Arithmetic Operations on Images

This includes bitwise AND, OR, NOT and XOR operations. They will be highly useful while extracting any part of the image.

Used to cut the image to the only interested region.



Color conversions

RGB ↔ HSV

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-min(R,G,B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images: V←255V,S←255S,H←H/2(to fit to 0 to 255)
- 16-bit images: (currently not supported) V<−65535V,S<−65535S,H<−H
- 32-bit images: H, S, and V are left as is

RGB ↔ HLS

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V_{max} \leftarrow max(R,G,B)$$

$$V_{min} \leftarrow min(R,G,B)$$

$$L \leftarrow \frac{V_{max}+V_{min}}{2}$$

$$S \leftarrow \begin{cases} \frac{V_{max}-V_{min}}{V_{max}+V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max}-V_{min}}{2-(V_{max}+V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V_{max}-V_{min}) & \text{if } V_{max}=R \\ 120+60(B-R)/(V_{max}-V_{min}) & \text{if } V_{max}=G \\ 240+60(R-G)/(V_{max}-V_{min}) & \text{if } V_{max}=B \end{cases}$$

If $H<0$ then $H \leftarrow H+360$. On output $0 \leq L \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255 \cdot V, S \leftarrow 255 \cdot S, H \leftarrow H/2$(to fit to 0 to 255)
- 16-bit images: (currently not supported) $V<-65535 \cdot V, S<-65535 \cdot S, H<-H$
- 32-bit images: H, S, V are left as is

They are Used in detecting yellow and white colors.

Histogram Calculation

It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

Canny:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

Erosion:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

Color conversions:

https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html