Object detection in video

Implementation of an Android application using OpenCV and YOLO

Matteo Medioli

July 2019

Intelligent Systems for Pattern Recognition
Università degli Studi di Pisa

Introduction

Object Detection vs Image Classification

Image Classification

Input image

⇓

1) class label associated with the image

2) confidence score associated with prediction

Object Detection

Input image

⇓

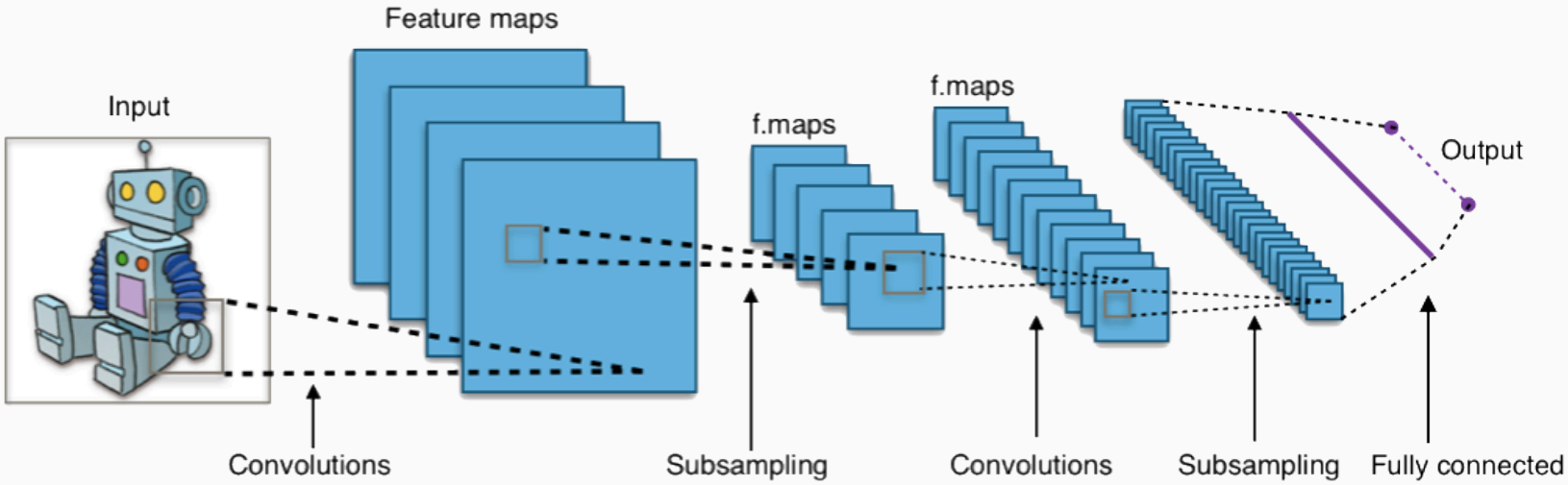
1) class label associated with each bounding box

2) confidence score associated with each bounding box and class label

3) a list of bounding boxes

Image Classification: deep learning approach

Classification through Convolutional Neural Network

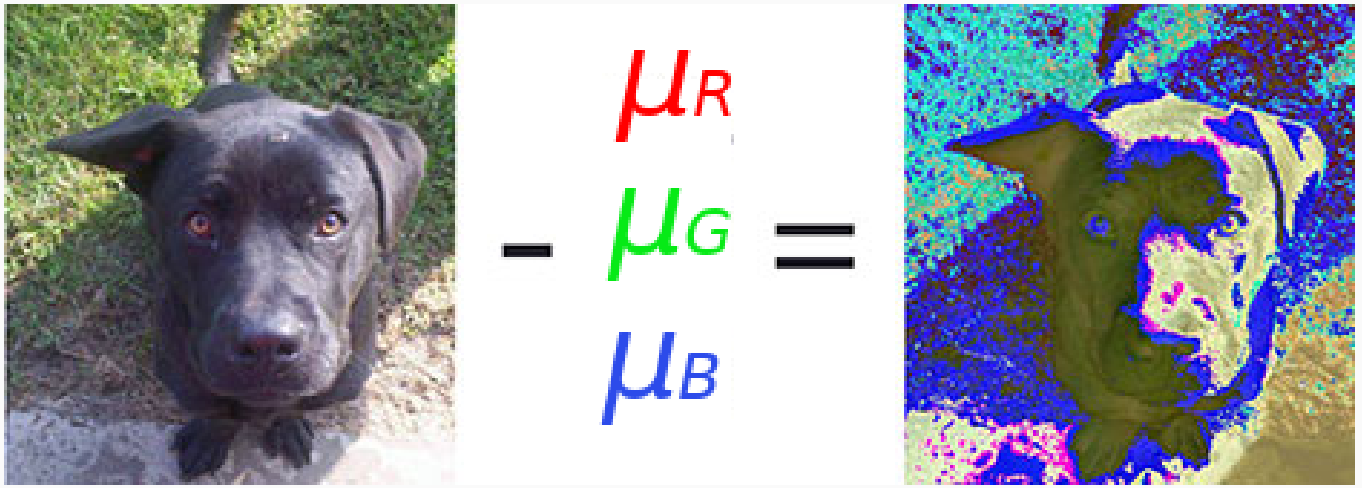


- **Convolutional layers**: vanishing or exploding gradients problem / curse of dimensionality
- **Pooling layers**: subsampling, reduce spatial size of representations, control overfitting
- **Fully connected layers**: classification task

Preprocessing: Mean subtraction & Scaling

Mean subtraction : used to help combat illumination changes in the input images in our dataset.

$$R = R - \mu_R \quad G = G - \mu_G \quad B = B - \mu_B$$



Scale factor : possible choice of scale factor σ to add normalization to data in the three levels.

$$R = \frac{(R-\mu_R)}{\sigma} \quad G = \frac{(G-\mu_G)}{\sigma} \quad B = \frac{(B-\mu_B)}{\sigma}$$

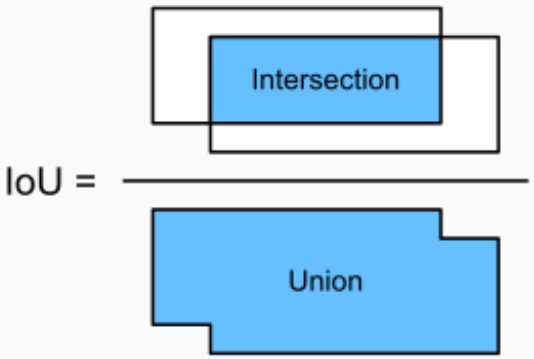
Object detection: what's the difference?

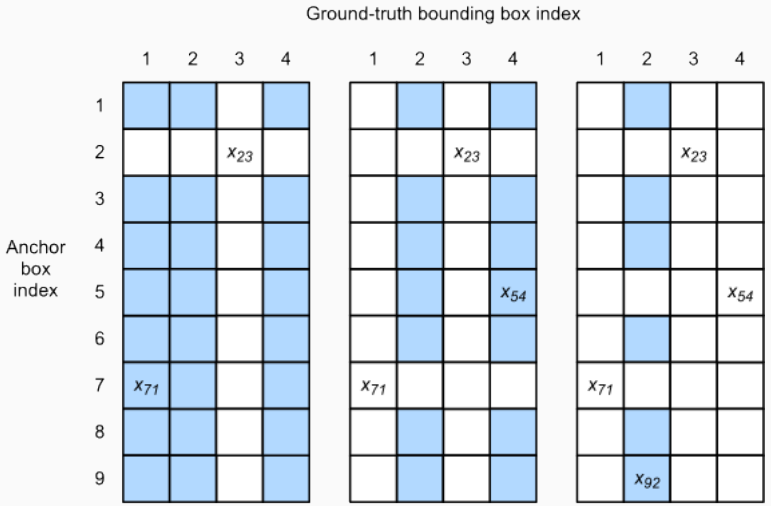
Need to also detect the position through bounding boxes:

Object detection algorithms usually sample a large number of regions in the input image, determine whether these regions contain objects of interest and adjust the edges of the regions so as to predict bounding boxes.

- n_A Anchor boxes $\mathcal{A}$
- n_B Ground-truth boxes $\mathcal{B}$

Intersect over Union (IoU) $\Longrightarrow J(\mathcal{A},\mathcal{B}) = \frac{|\mathcal{A}\cap\mathcal{B}|}{|\mathcal{A}\cup\mathcal{B}|}$





Non Maximum Suppression

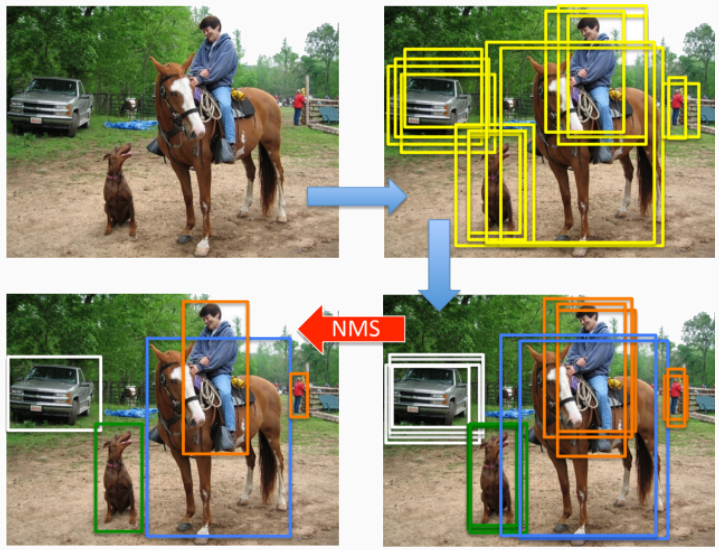
Model prediction phase:

1. generate multiple anchor boxes for the image
2. set background threshold based on IoU
3. predict categories and offsets for these anchor boxes one by one
4. obtain prediction labeled bounding boxes based on anchor boxes

Multiple proposals for single object: each output prediction is $\begin{bmatrix} b_x \\ b_y \\ b_w \\ b_h \\ p_c \end{bmatrix}$

Algorithm:

1. Discard all boxes with $p_c < NMS_{thr}$
2. while (remainBoxes())
   - 2.1 pick box B_{high} with **highest** p_c
   - 2.2 set B_{high} as a **true prediction**
   - 2.3 Discard all remaining box B_j s.t **IoU**$(B_j, B_{high}) \geq 0.5$



YOLO: You Only Look Once

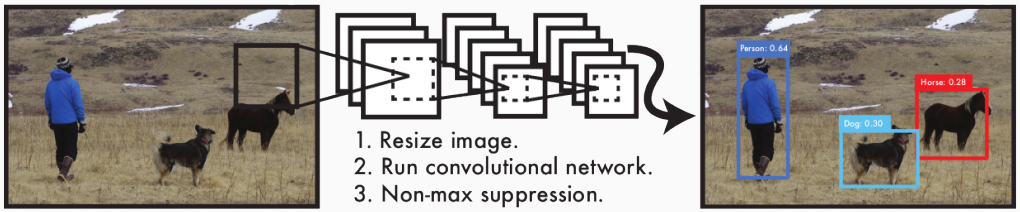
YOLO approach

Prior work (sliding window, region proposal-based...)

- divide image in regions
- apply one classifier on all image regions

YOLO [1]

Object detection as a **regression problem to spatially separated bounding boxes and associated class probabilities**.
A single neural network predicts bounding boxes and class probabilities directly from **full images in one evaluation**.



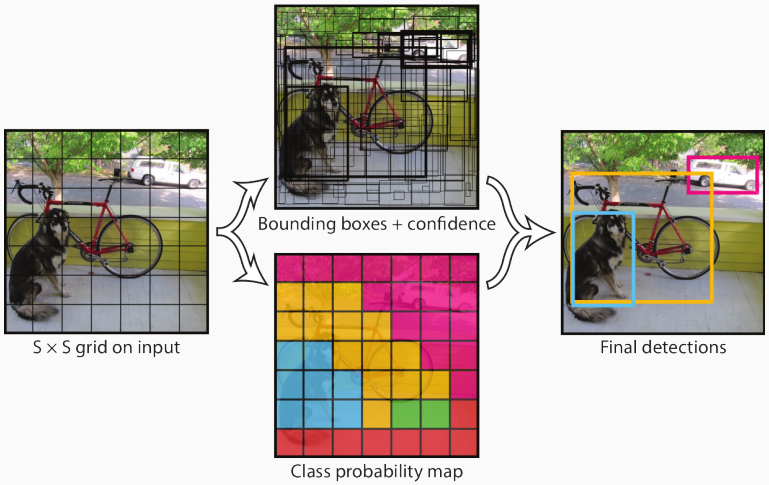
[1] [RF15]

Consequences of regression problem

1. **Fast model**
   Don't need a complex pipeline. We simply run our neural network on a new image at test time.
   Real time object detection in video

2. **Unified Detection**: YOLO reasons globally
   Box confidence score $\Leftarrow$ **IoU(B)** (not yet anchor box)
   Object confidence score $\Leftarrow$ **class probability**

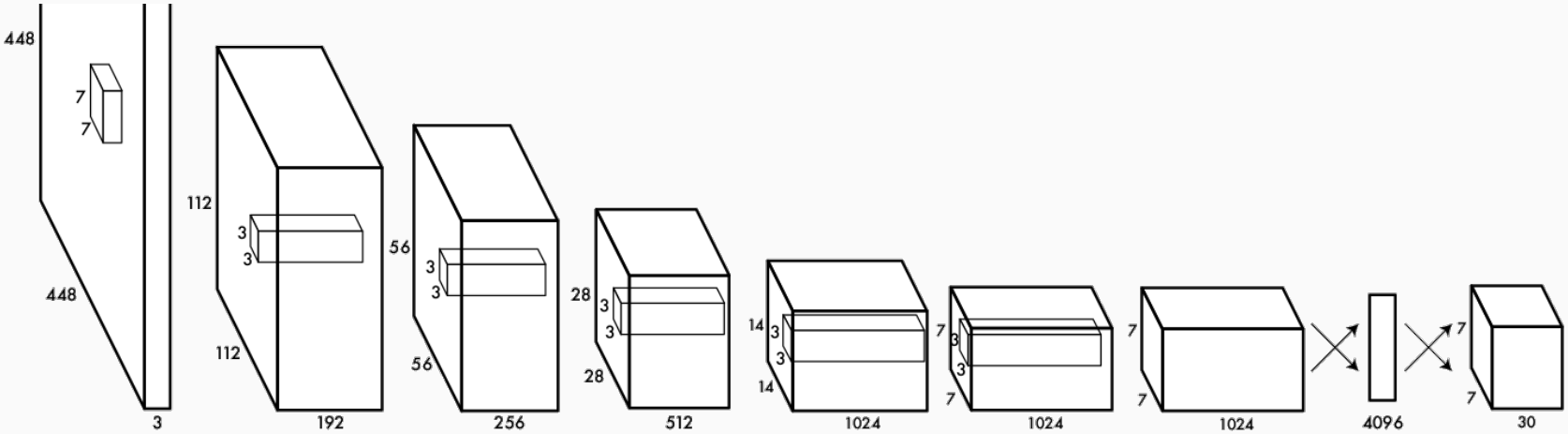
   Unified confidence score
   $Pr(Class_i|Object)Pr(Object) * IoU(B) = Pr(Class_i) * IoU(B)$



Divides the image into an **S × S grid** and for each grid cell predicts **B bounding boxes**, confidence for those boxes, and **C class probabilities**. These predictions are encoded as an **S × S × (B * 5 + C)** tensor.

Network Design

- 24 convolutional layers
  - 3x3 convolutional layers
  - 1x1 reduction layers
- Maxpool layers 2x2
- 2 fully connected layer
- Leaky ReLU $f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$



YOLOv2[2] & YOLOv3[3]: what's new?

YOLO v2

- Batch Normalization
- High Resolution Classifier
- Anchor Boxes
- Clustering box dimensions (k-means)
- Multi-Scale Training
- Hierarchical classification

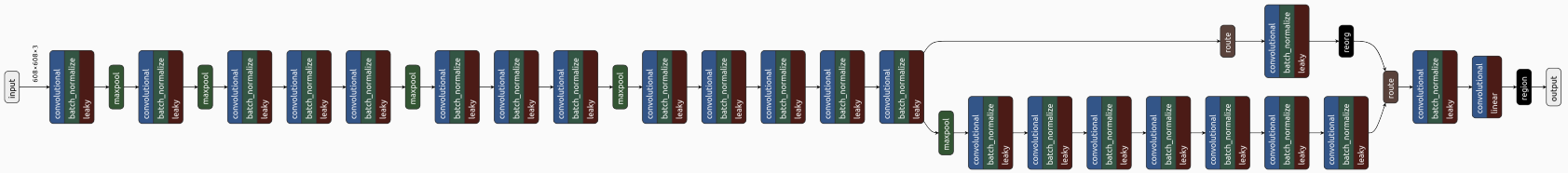
YOLO v3

- Logistic regression for bounding box score
- Multilabel
- Feature Extractor (53 conv. layers)

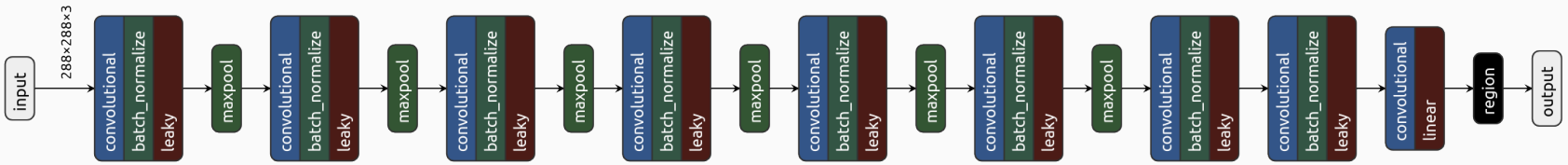
[2] [RF16]
[3] [RF18]

Darknet and Tiny YOLO

YOLOv2

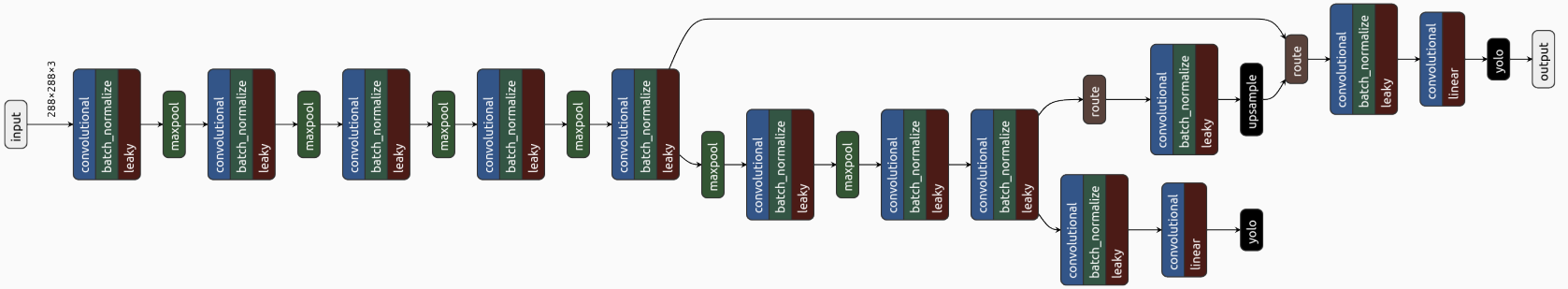


Tiny-YOLOv2



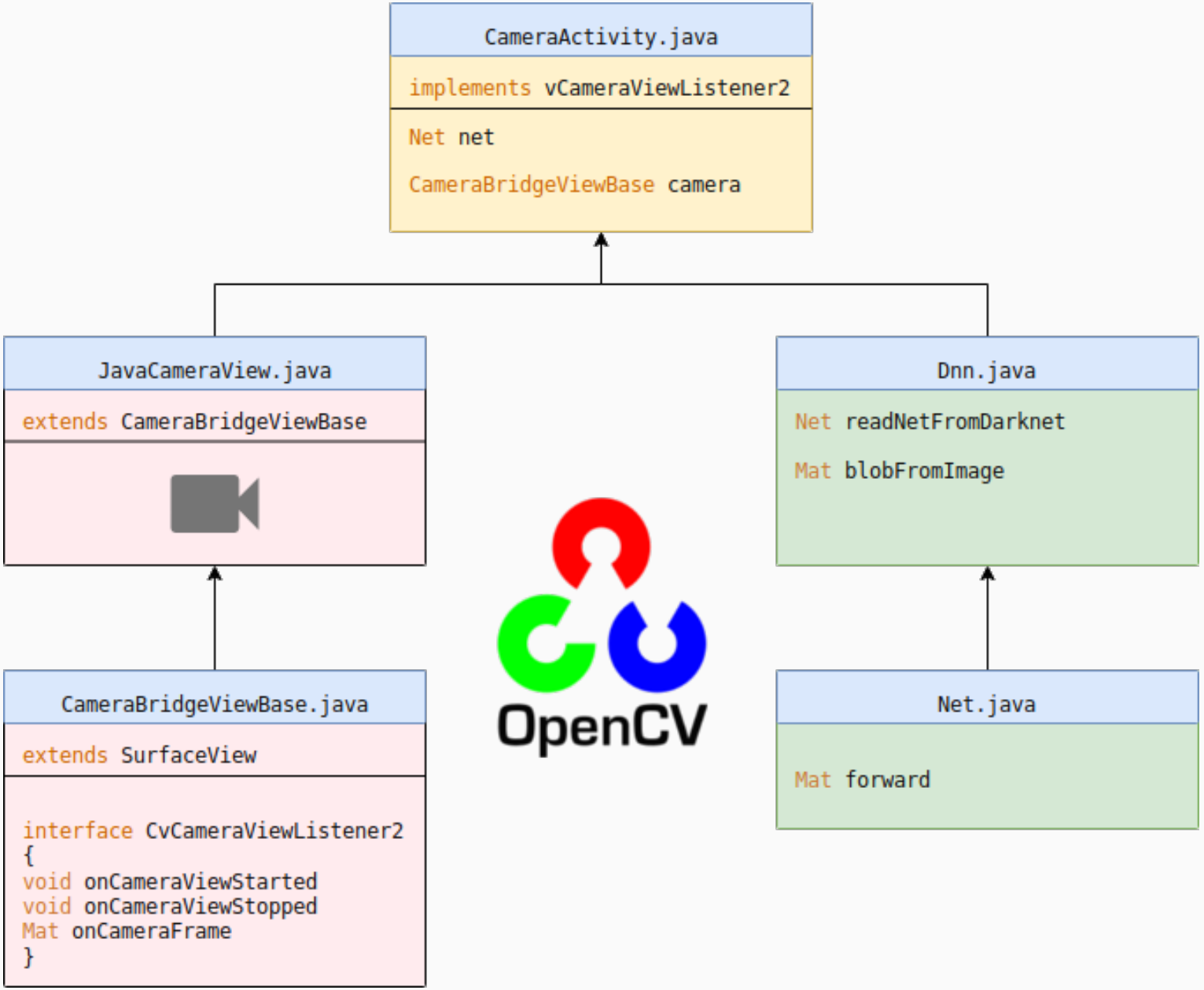
YOLOv3

Tiny-YOLOv3



Implementation

OpenCV: Java SDK and DNN module



Dnn: load network

Load convolutional network from *.cfg and *.weights files and read COCO Dataset labels in assets folder.

Framework: Darknet[4]

```
@Override
public void onCameraViewStarted(int width, int height) {
  String modelConfiguration = getAssetsFile("yolov2-tiny.cfg", this);
  String modelWeights = getAssetsFile("yolov2-tiny.weights", this);
  classNames = readLabels("labels.txt", this);
  net = Dnn.readNetFromDarknet(modelConfiguration, modelWeights);
}
```

[4] [dar]

Detection from camera preview

Mat onCameraFrame(CvCameraViewFrame inputFrame)

Three phases:

- Preprocessing frame preview

```
@Override
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    Mat frame = inputFrame.rgba();
    Imgproc.cvtColor(frame, frame, Imgproc.COLOR_RGBA2RGB);
    Size frameSize = new Size(288, 288);
    Scalar mean = new Scalar(127.5);
    Mat blob = Dnn.blobFromImage(frame, 1.0 / 255.0, frameSize, mean);
    net.setInput(blob);
```

- Forward pass for detection

```
    List<Mat> result = new ArrayList<>();
    List<String> outBlobNames = net.getUnconnectedOutLayersNames();
    net.forward(result, outBlobNames);
```

Retrieve and draw detections

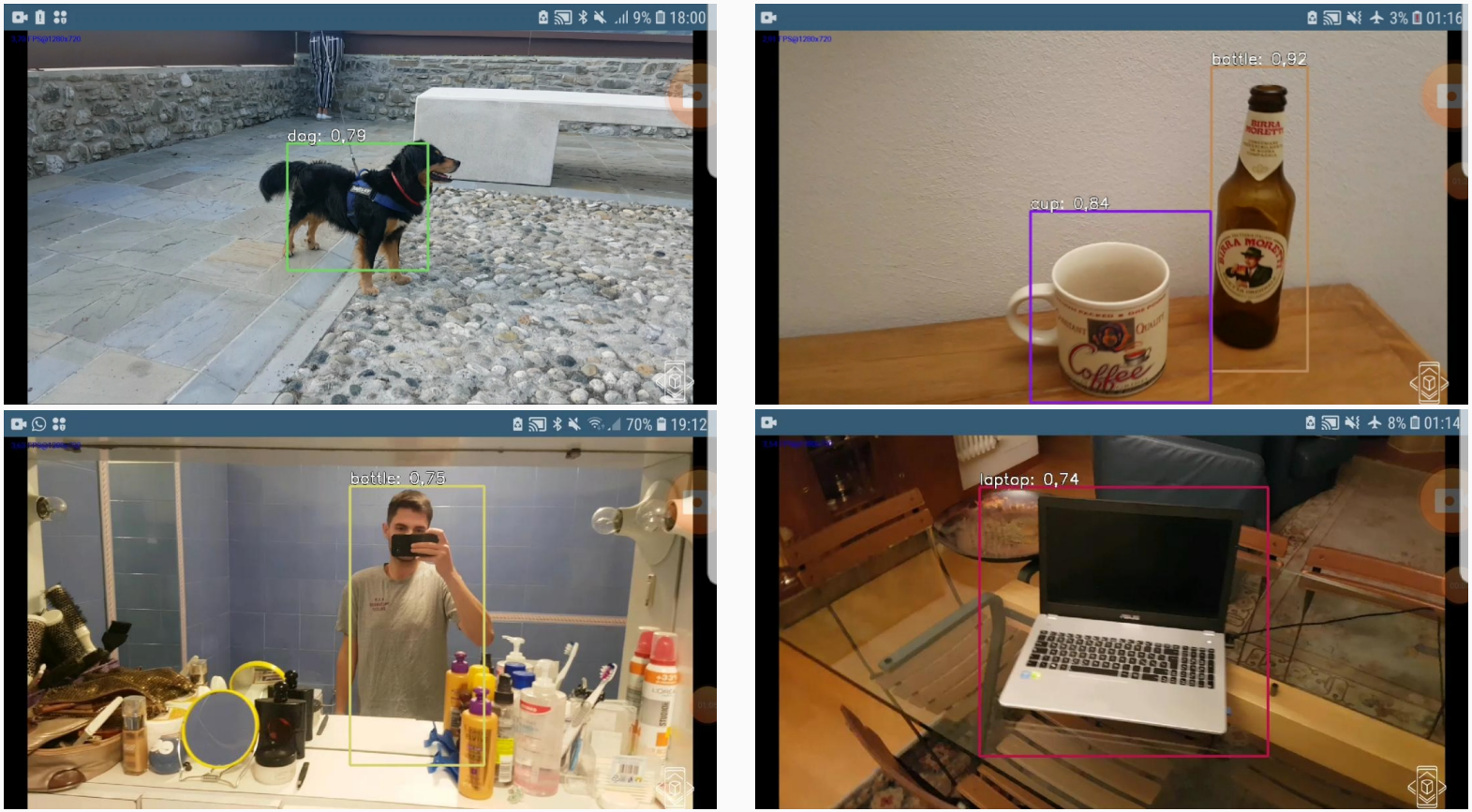
- Retrieve from List<Mat> result

```
float confThreshold = 0.5f;
 // each result element is a different located obj.
for (int i = 0; i < result.size(); ++i) {
   Mat level = result.get(i);
   // each level row is a candidate detection
   for (int j = 0; j < level.rows(); ++j) {
      Mat row = level.row(j);
      Mat scores = row.colRange(5, level.cols());
      Core.MinMaxLocResult mm = Core.minMaxLoc(scores);
      float confidence = (float) mm.maxVal;
      Point classIdPoint = mm.maxLoc;
      if (confidence > confThreshold) {
         // get box coordinates
         int centerX = (int) (row.get(0, 0)[0] * frame.cols());
         int centerY = (int) (row.get(0, 1)[0] * frame.rows());
         int width = (int) (row.get(0, 2)[0] * frame.cols());
         int height = (int) (row.get(0, 3)[0] * frame.rows());
         int left = (int) (centerX - width * 0.5);
         int top =(int)(centerY - height * 0.5);
         int right =(int)(centerX + width * 0.5);
         int bottom =(int)(centerY + height * 0.5);
         // Draw and labeling with Imgproc.java
```

Results and improvements

Results

Screenshots



Video examples

▶ Cars ▶ Table ▶ DogMan

Considerations

- Reached discrete detections and a decent FPS ratio even without using GPUs
- Better to use yolov2-tiny as it is less deep, giving up precision
- By resizing the input to a minimum size (288x288), however, good detection results are achieved
- OpenCV makes it easy to switch models (Caffe, Tensorflow, Torch) making this implementation really versatile
- To have a high performance real time object detection YOLO is the fastest usable, but running on Android it is necessary to apply many improvements to the application.

Improvements

- GPU support and Hardware Acceleration
- CameraX to JavaCameraView
- Add choice to run different DL models
- Portrait mode

Check project repository at
`https://github.com/Ickarus/AndroidObjectDetection-OpenCV`

References i

- *Darknet: Open source neural networks in c*, `hhttps://pjreddie.com/darknet/`, Redmon, Joseph.
- Joseph Redmon and Ali Farhadi, *You only look once: Unified, real-time object detection*, arXiv (2015).
- _____, *Yolo9000: Better, faster, stronger*, arXiv (2016).
- _____, *Yolov3: An incremental improvement*, arXiv (2018).