Technical Overview of a YOLOX C++ Inference Pipeline

Your Name

July 25, 2025

Abstract

This document presents a detailed, step-by-step explanation of a C++ inference pipeline for the YOLOX object detection model, from image pre-processing to final visualization. Code snippets illustrate each phase, enabling reproduction in C++ using OpenCV and ONNX Runtime.

1 Introduction

We describe here the end-to-end flow:

1. Load and store the original image.
2. Pre-process via "letterbox" resize and pad.
3. Convert color space, normalize, and pack into the model's required memory layout.
4. Run ONNX Runtime inference.
5. Decode raw network outputs into bounding boxes.
6. Map boxes back to original image coordinates.
7. Perform non-maximum suppression.
8. Draw final boxes and labels.

2 1. Image Loading

Load the image with OpenCV:

```
cv::Mat orig = cv::imread(image_path);
if(orig.empty()) throw std::runtime_error("Cannot load " + image_path);
int W_orig = orig.cols, H_orig = orig.rows;
```

3 2. Letterbox Pre-processing

Resize and pad so the image fits the network input without distorting aspect ratio:

$$r = \min\Big(\frac{W_{\text{in}}}{W_{\text{orig}}}, \frac{H_{\text{in}}}{H_{\text{orig}}}\Big), \quad W' = \lfloor r\, W_{\text{orig}} \rfloor,\ H' = \lfloor r\, H_{\text{orig}} \rfloor$$

```
float r = std::min(W_in/(float)W_orig, H_in/(float)H_orig);
int Wp = int(W_orig * r), Hp = int(H_orig * r);
cv::Mat tmp, letterboxed(H_in, W_in, orig.type(), cv::Scalar(114,114,114));
cv::resize(orig, tmp, cv::Size(Wp,Hp));
tmp.copyTo(letterboxed(cv::Rect(0,0,Wp,Hp)));
```

4 3. Color Conversion, Normalization, and Layout Packing

Convert BGR→RGB, scale to $[0, 1]$, and reorganize from HWC to CHW:

```
cv::Mat rgb, f32;
cv::cvtColor(letterboxed, rgb, cv::COLOR_BGR2RGB);
rgb.convertTo(f32, CV_32F, 1/255.0f);
std::vector<cv::Mat> chs;
cv::split(f32, chs);
std::vector<float> blob;
blob.reserve(3*H_in*W_in);
for(int c=0;c<3;++c){
  float* ptr = (float*)chs[c].data;
  blob.insert(blob.end(), ptr, ptr + H_in*W_in);
}
```

5 4. ONNX Runtime Inference

Wrap the blob in an Ort tensor and run:

```
Ort::Value inputTensor = Ort::Value::CreateTensor<float>(
  memory_info_, blob.data(), blob.size(),
  input_shape.data(), input_shape.size()
);
auto outputs = session->Run(
  Ort::RunOptions{nullptr},
  &inputName, &inputTensor, 1,
  &outputName, 1
);
```

6 5. Decoding YOLOX Output

Each output row $\mathbf{y} = [\Delta x, \Delta y, \Delta w, \Delta h, \text{obj}, s_1, \ldots, s_C]$ is decoded as:

$$\begin{aligned} x_p &= (i + \sigma(\Delta x)) \cdot \text{stride}, \quad y_p = (j + \sigma(\Delta y)) \cdot \text{stride}, \\ w_p &= e^{\Delta w} \cdot \text{stride}, \quad h_p = e^{\Delta h} \cdot \text{stride}, \\ \text{score} &= \sigma(\text{obj}) \times \max_c \sigma(s_c), \end{aligned}$$

and filtered by score $\geq \tau$. In code:

```
for each grid cell (i,j):
  float dx = b[0], dy=b[1], dw=b[2], dh=b[3], obj=b[4];
  float cx = (i + sigmoid(dx))*stride;
  float cy = (j + sigmoid(dy))*stride;
  float w  = exp(dw)*stride, h = exp(dh)*stride;
  int bestC = argmax_c sigmoid(b[5+c]);
  float score = sigmoid(obj)*sigmoid(b[5+bestC]);
  if(score < confThresh) continue;
  // store detection
```

7 6. Undo Letterbox Back to Original Coordinates

Mapped corners from padded $(W_{\text{in}}, H_{\text{in}})$ to $(W_{\text{orig}}, H_{\text{orig}})$:

$$x_1 = \frac{x_{1p}}{r}, \quad y_1 = \frac{y_{1p}}{r}, \quad x_2 = \frac{x_{2p}}{r}, \quad y_2 = \frac{y_{2p}}{r}.$$

```
x1 = std::clamp((x1p)/r, 0.f, W_orig);
y1 = std::clamp((y1p)/r, 0.f, H_orig);
x2 = std::clamp((x2p)/r, 0.f, W_orig);
y2 = std::clamp((y2p)/r, 0.f, H_orig);
```

8 7. Non-Maximum Suppression (NMS)

Greedily suppress overlapping boxes of the same class:

```
std::sort(dets.begin(), dets.end(),
          [](auto &a,auto &b){return a.score>b.score;});
for(size_t i=0; i<dets.size(); ++i){
  if(removed[i]) continue;
  keep.push_back(dets[i]);
  for(size_t j=i+1; j<dets.size(); ++j)
    if(!removed[j] && IoU(dets[i],dets[j])>nmsThresh)
      removed[j] = true;
}
```

9 8. Final Visualization

Draw surviving boxes and labels on the original image:

```
for(auto &d:finalDetections){
  cv::rectangle(orig,
    cv::Point(d.x1,d.y1), cv::Point(d.x2,d.y2),
    cv::Scalar(0,255,0),2);
  cv::putText(orig, label,
    cv::Point(d.x1,d.y1-4),
    cv::FONT_HERSHEY_SIMPLEX,0.5,
    cv::Scalar(0,0,0),1);
}
cv::imwrite("annotated.jpg", orig);
```

A List of Abbreviations

IoU Intersection over Union

NMS Non-Maximum Suppression

RGB Red–Green–Blue

BGR Blue–Green–Red

ONNX Open Neural Network Exchange

ORT ONNX Runtime