Secure Multiparty Computation Meets Deep Learning

Yoshi234

2024-01-30

Table of contents

Preface

This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

Contents

The quarto book contains an organized structure of notes on a variety of secure multiparty computation protocols, implementation details, and a discussion of v2x applications and relevant deep learning models. The authors hope that any readers find these resources useful.

Resources

The matcha editor is used to construct some of the mathematical diagrams shown in this book. In order to export a matcha diagram, you need to enter the full-screen mode for the diagram, and click on the "export" drop-down which becomes available. This will allow you to save the math diagram as a png image. We will make liberal use of these throughout the book, especially for explaining complex security primitives such as beaver's triples and homomorphic encryption.

Part I

Secure Multiparty Computation Primitives

This section of the text discusses a number of secure multiparty computation protocols in-depth and provides sample implementation to get you started.

1 Beaver's Triples Explained

In order to mask fixed-point multiplication inputs, we need to use a secure primitive known as Beaver's triples. These were originally introduced by Beaver (1991) in his paper: Efficient Multiparty Protocols Using Circuit Randomization

1.1 Generating Beaver's Triples

The first step in masking multiplications is to generate Beaver's Triples. These are random values a, b, c such that $a \cdot b = c$

1.1.1 Step 1: Generating Inputs

Suppose we have two parties, **Alice** and **Bob**. Here, we will suppose that Alice is the sender, and Bob is the receiver. All this really means is that Alice sends her values to Bob first, with Bob sending his values to Alice thereafter.

We begin by first generating a random pair (a_i, b_i) for each party where i corresponds to the party in question. We generate these from the field F_2^2 which simply means that we select two random bits from $\{0, 1\}$

$$\begin{array}{ccc} \text{Alice} & & \text{Bob} \\ \mathbb{F}_2^2 \xrightarrow{\$} (a_A, b_A) & \overset{\&}{\rightsquigarrow} & \mathbb{F}_2^2 \xrightarrow{\$} (a_B, b_B) \end{array}$$

Next, Bob and Alice each generate random masking values r_A and r_B respectively.

$$\begin{array}{cc} \text{Alice} & \text{Bob} \\ \mathbb{F}_2 \xrightarrow{\$} r_A & \mathbb{F}_2 \xrightarrow{\$} r_B \end{array}$$

1.1.2 Step 2: Sending and Receiving Shares

After these masking values are generated, Alice uses oblivious transfer (see Chapter 2) to send $(r_A, r_A \oplus a_A)$ to Bob.

When Bob receives this, he obtains $b_B a_A \oplus r_A$ and then sends $(r_B, r_B \oplus a_B)$ to Alice

$$\begin{array}{ccc} \text{Alice} & & \text{Bob} \\ \text{Snd } = (r_A, r_A \oplus a_A) & \xrightarrow{OT} & \text{Rcv } = b_B a_A \oplus r_A \\ \text{Rcv } = b_A a_B \oplus r_B & \xleftarrow{OT} & \text{Snd } = (r_B, r_B \oplus a_B) \end{array}$$

After receiving these shares, Alice and Bob each received shares from one another, they `XOR` the product **(intersection)** of their shares with the ones received from the other party

$$\begin{array}{cc} \text{Alice} & \text{Bob} \\ c_A : \overbrace{r_A \oplus a_A b_A}\ \text{Alice's og share} \oplus b_A a_B \oplus r_B & c_B : \overbrace{r_B \oplus a_B b_B}\ \text{Bob's og share} \oplus b_B a_A \oplus r_A \\ \downarrow & \downarrow \\ c_A : \text{—} r_A \text{—} \oplus a_A b_A \oplus b_A a_B \oplus \text{—} r_B \text{—} & \\ c_B : \text{—} r_B \text{—} \oplus a_B b_B \oplus b_B a_A \oplus \text{—} r_A \text{—} & \end{array}$$

1.1.3 Step 3: Final Simplification

Now that each party has obtaiined a preliminary share of C, C_A and C_B possessed by Alice and Bob respectively, we take the `XOR` of the two components to get the product $a \cdot b = c$

First, let us simplify c_A and c_B a bit more

$$c_A = a_A b_A \oplus b_A a_B a_A \oplus a_B = a b_A \oplus b_A = b_A c_B = a_B b_B \oplus b_B a_A a_B \oplus a_B = a b_B \oplus b_B = b_B c_A = a(b_A) c_B = a(b_B)$$

Now that we have simplified these terms, we can compute the final result

$$c = c_A \oplus c_B c = a(b_A) \oplus a(b_B) c = a(b_A \oplus b_B) c = a \cdot b$$

Thus, we have shown how beaver's triples are randomly generated to achieve the desired output.

2 Oblivious Transfer

Part II

Deep Learning

This portion of the text deals with deep learning and artificial intelligence applications. Specifically, it covers topics such as object detection and classification with computer vision, neural network primitives, and others. Other machine learning applications and import resources may also be included in this portion of the text

3 Residual connections

This section discusses residual connections based on the information provided by Wong (2021) in his medium article

3.1 Motivation for Residual connections

Deep Neural Networks such as YOLO (see Section 4.2.4.3) allow for greater accuracy and performance. However, deep networks like this make it more difficult for the model to converge during training.

Residual connections help to make training networks easier.

3.2 Formulation

Residual connections - allow data to reach other parts of a sequential network by skipping layers

This flow is depicted well by the following image:

Steps:

1. Apply identity mapping to x - perform element-wise addition $F(x) + x$: this is the **residual block**
    1. residual blocks may also include a ReLU actiation applied to $F(x)+x$. *This works when dimensions of $F(x)$ and x are the same
    2. If dimensions of $F(x)$ and x are NOT the same, then you can multiply x by some matrix of constants W to scale it. $F(x) + Wx$

3.3 The Utility of Residual Blocks for Training Deep Networks

Empirical results have demonstrated that residual blocks increase the speed and ease of network convergence. There are a number of suspected reasons as to why this enables such performance gains.

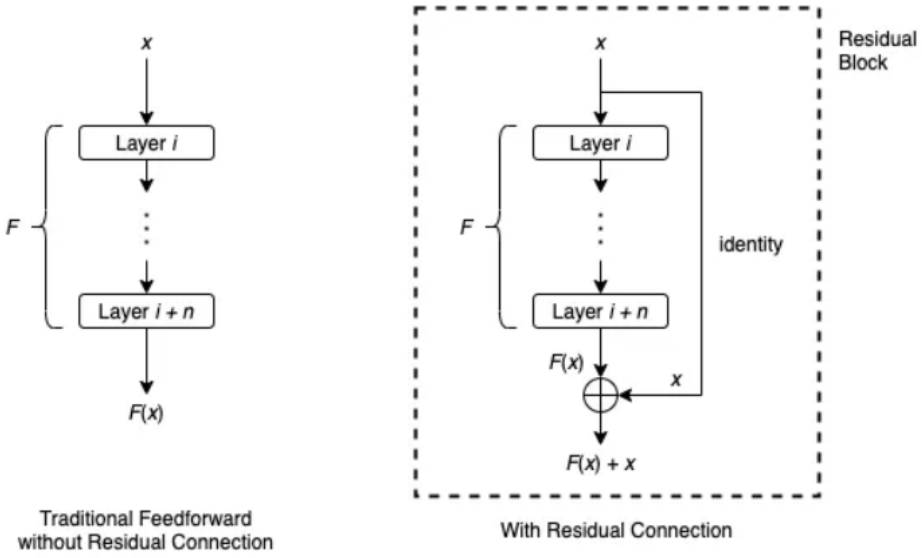

Figure 3.1: residual connection diagram

3.3.1 Ensemble of Shallow Neural Networks

3.3.2

4 Object Detection Algorithms

4.1 SSD: Single Shot MultiBox Detector

The single shot multibox detector is an algorithm presented by W. Liu (2016) for the purpose of taking a 300 x 300 input and generating bounding boxes on objects of interest within the image. The paper is linked here

4.2 A Comprehensive Review of YOLO (v1 to v8+)

J. Terven (2023) present review and analysis of the evolution of the Yolo algorithm, with a focus on the innovations and contributions made by each iteration, as well as the major changes in network architecture (and training tricks) which have been implemented over time.

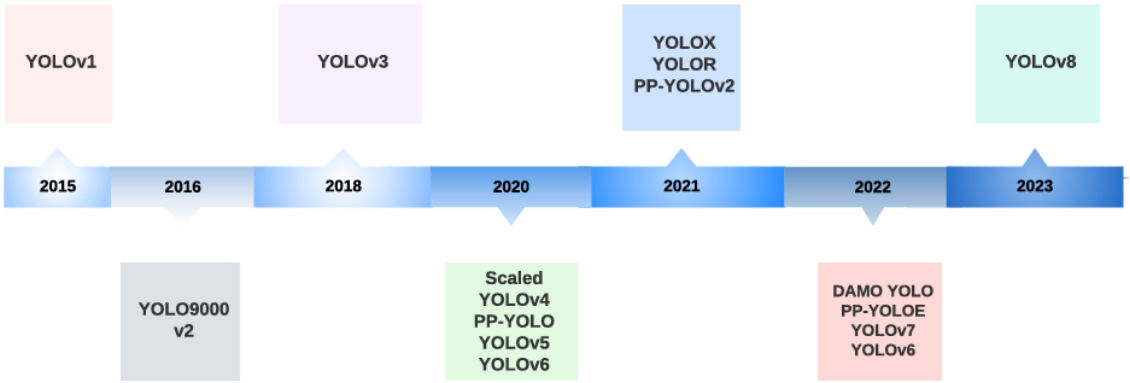

Figure 1: A timeline of YOLO versions.

Figure 4.1: A timeline of YOLO development

4.2.1 Applications of YOLO

Yolo has proven invaluable for a number of different applications

- autonomous vehicles
  - enables quick identification and tracking of objects like vehicles, pedestrians, bicycles and other obstacles
- action recognition
- video surveillance
- sports analysis
- human-computer interaction
- crop, disease, pest detection and classsification
- face detection - biometrics, security, facial recognition
- cancer detection
- skin segmentation
- pill identification
- remote sensing
  - satellite and aerial imagery object detection / classification
  - land use mapping
  - urban planning
- security systems
- smart transportation systems
- robotics and drones

4.2.2 Evaluation Metrics

Average Precision (AP) and Mean Average Precision (mAP) are the most common metrics used in the object detection task. It measures average precision across all categories, providing a single value to compare different models

4.2.2.1 How AP works

- mAP is the average precision for accuracy of predictions across all classes of objects contained within an image
  - individual AP values are determined for each category separately.
- IOU (intersection over union)
  - measures the proportion of the predicted bounding box which overlaps which overlaps with the true bounding box

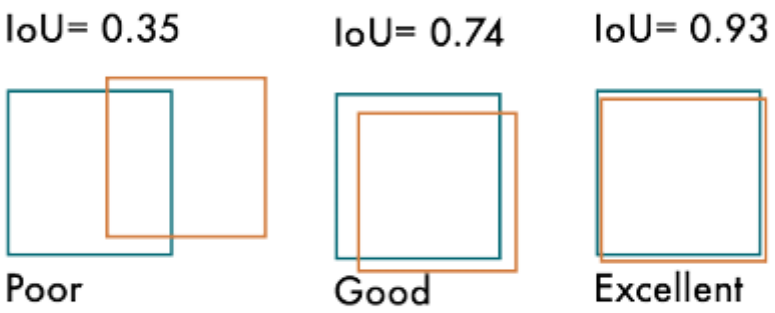

Figure 4.2: Intersection over union in practice

Different methods are used to compute AP when evaluating object detection methods on the COCO and VOC datasets (PASCAL-VOC)

4.2.3 Non-Maximum Suppression

A post-processing technique - reduces number of overlapping boxes and improves detection quality. Object detectors typically generate multiple bounding boxes around the same object. Non-max suppression picks the best ones and gets rid of the others.

The algorithm for this is defined below:

Algorithm 1 Non-Maximum Suppression Algorithm

Require: Set of predicted bounding boxes B, confidence scores S, IoU threshold τ, confidence threshold T
Ensure: Set of filtered bounding boxes F

$F \leftarrow \emptyset$
Filter the boxes: $B \leftarrow \{b \in B \mid S(b) \geq T\}$
Sort the boxes B by their confidence scores in descending order
while $B \neq \emptyset$ **do**
    Select the box b with the highest confidence score
    Add b to the set of final boxes F: $F \leftarrow F \cup \{b\}$
    Remove b from the set of boxes B: $B \leftarrow B - \{b\}$
    for all remaining boxes r in B **do**
        Calculate the IoU between b and r: $iou \leftarrow IoU(b, r)$
        if $iou \geq \tau$ **then**
            Remove r from the set of boxes B: $B \leftarrow B - \{r\}$
        end if
    end for
end while

Figure 4.3: Non-Max Suppression Alg

A useful visualization is also provided:

Figure 4.4: Non-Max Supression Vis

4.2.4 YOLO

The original authors of YOLO titled it as such for the reason that it only required a single pass on the image to accomplish the detection task. This is contrast to the other approaches used by Fast R-CNN and sliding window methods.

The output coordinates of the bounding box were detected using more straightforward regression techniques

4.2.4.1 YOLOv1

PASCAL-VOC AP: 63.4%

YOLOv1 predicted all bounding boxes simultaneously by the following process:

1. divide image into $S \times S$ grid
2. predict B bounding boxes of the same class and confidence for C different classes per grid element
3. each bounding box had five values:
    1. Pc - confidence score for the bounding box - how likely it contains an object and the accuracy of the box
    2. bx and by - coordinates of center of box relative to grid cell.
    3. bh and bw - height and width of box relative to full

4. output an $S \times S \times (B \times 5 + C)$ tensor
5. (optional) NMS used to remove redundant bounding boxes

Here is an example of that output:

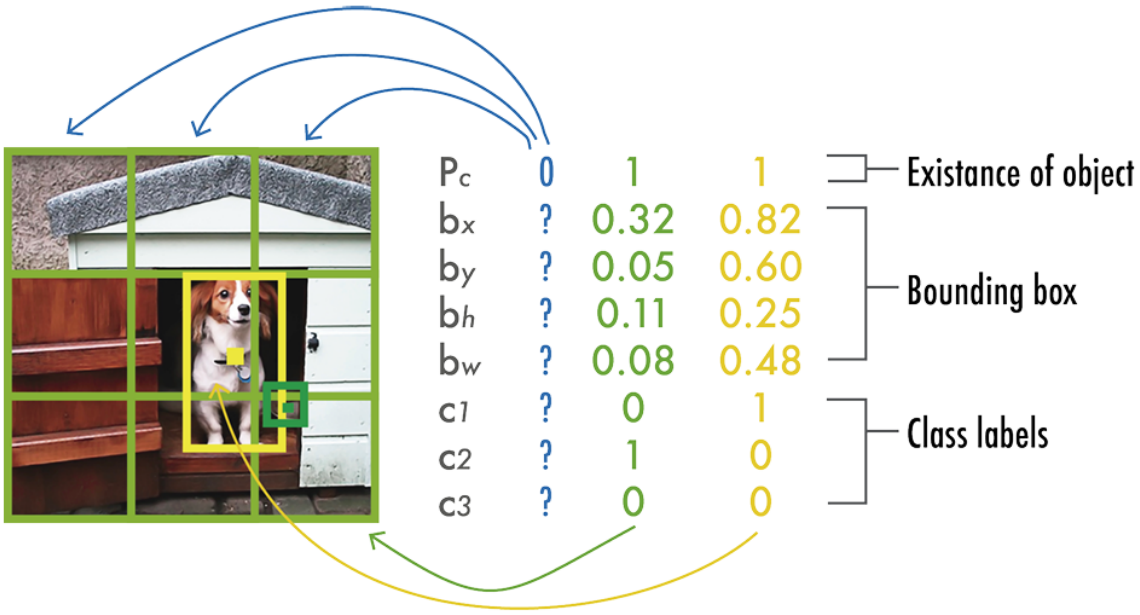

Figure 4.5: yolo output prediction

4.2.4.1.1 v1 Architecture

Normal Architecture

- 24 conv layers
  - 1×1 conv layers are used - reduce number of feature maps and keep parameters lower
  - leaky rectified linear unit activations
- 2 fc layers
  - predict bounding box coordinates / probs
  - linear activation function for final layer

FastYOLO

- Used 9 conv layers instead of 24 for greater speed (at the cost of reduced accuracy)

	Type	Filters	Size/Stride	Output
	Conv	64	7×7 / 2	224×224
	Max Pool		2×2 / 2	112×112
	Conv	192	3×3 / 1	112×112
	Max Pool		2×2 / 2	56×56
$1\times$	Conv	128	1×1 / 1	56×56
	Conv	256	3×3 / 1	56×56
	Conv	256	1×1 / 1	56×56
	Conv	512	3×3 / 1	56×56
	Max Pool		2×2 / 2	28×28
$4\times$	Conv	256	1×1 / 1	28×28
	Conv	512	3×3 / 1	28×28
	Conv	512	1×1 / 1	28×28
	Conv	1024	3×3 / 1	28×28
	Max Pool		2×2 / 2	14×14
$2\times$	Conv	512	1×1 / 1	14×14
	Conv	1024	3×3 / 1	14×14
	Conv	1024	3×3 / 1	14×14
	Conv	1024	3×3 / 2	7×7
	Conv	1024	3×3 / 1	7×7
	Conv	1024	3×3 / 1	7×7
	FC		4096	4096
	Dropout 0.5			4096
	FC		$7 \times 7 \times 30$	$7 \times 7 \times 30$

Figure 4.6: yolo v1 architecture

4.2.4.1.2 v1 Training

Basic training process:

1. pretrain first 20 layers at resolution 224×224 with *ImageNet* dataset
2. add last four layers with randomly initialized weights - fine tune model with PASCAL VOC 2007 and PASCAL VOC 2012 at resolution 448×448

Loss functions:

- scaling factors
  - $\lambda_{coord} = 5$ - gives more weight to boxes with objects
  - $\lambda_{noobj} = 0.5$ - reduces importance of boxes with no object
- localization loss:
  - first two terms
  - computes error in predicted bounding box locations (x, y) and (w, h)
  - only penalizes boxes with objects in them
- confidence loss:
  - confidence error when object is detected (third term)
  - confidence error when no object is in box (fourth term)
- classification loss:
  - squared error of class conditional probabilities for each class if an object appears in the cell

4.2.4.2 YOLOv2 (YOLO 9000)

PASCAL-VOC AP: 78.6%

Improvements / Changes

1. Batch normalization - included on all convolutional layers
2. Higher resolution classifier - pretrained model (224 x 224) and then fine-tuned with ImageNet at a higher reoslution (448 x 448) for ten epochs
3. fully convolutional - remove dense layers and use fully conv architecture
4. use anchor boxes to predict bounding boxes
   1. anchor box - box with predefined shapes for prototypical objects
   2. defined for each grid cell
   3. system predicts coordinates and class for every anchor box

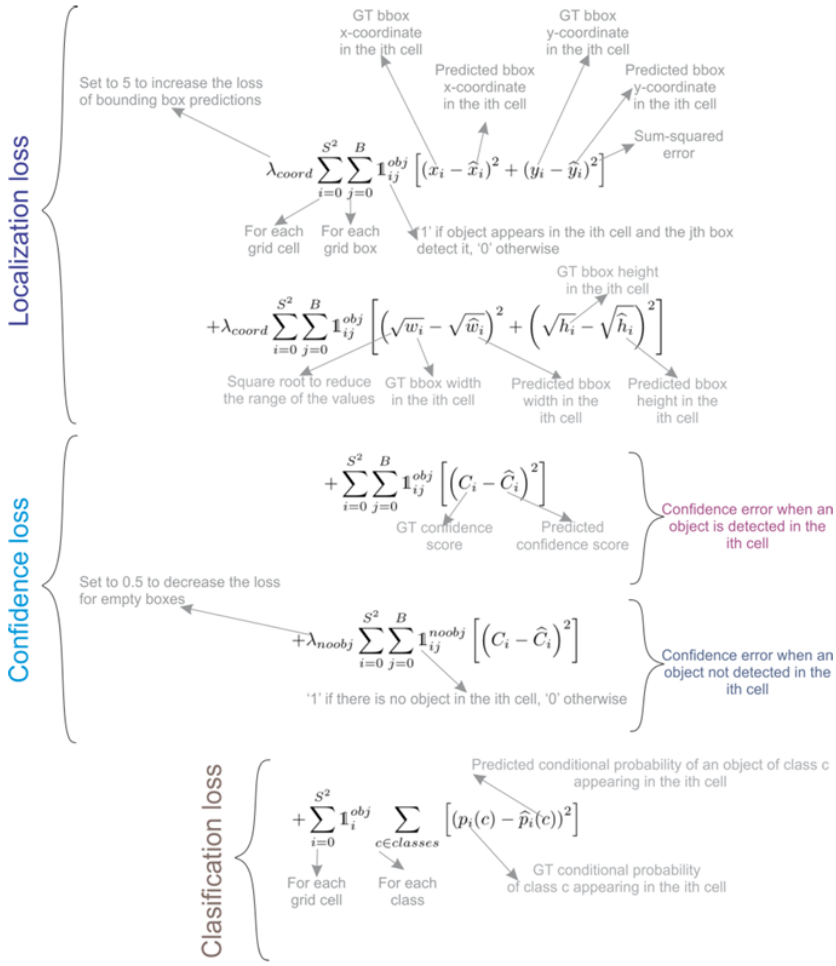

Figure 4.7: yolo v1 loss function

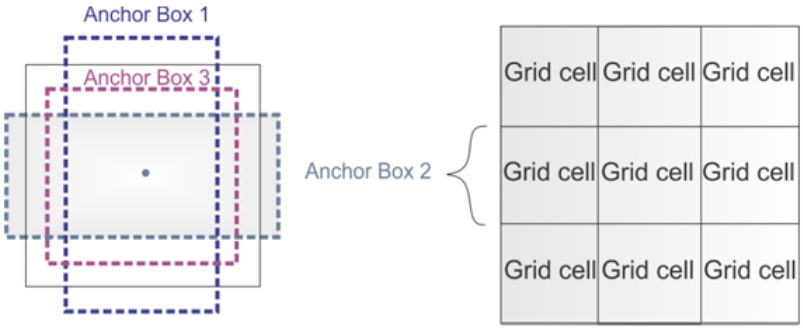

Figure 6: Anchor boxes. YOLOv2 defines multiple anchor boxes for each grid cell.

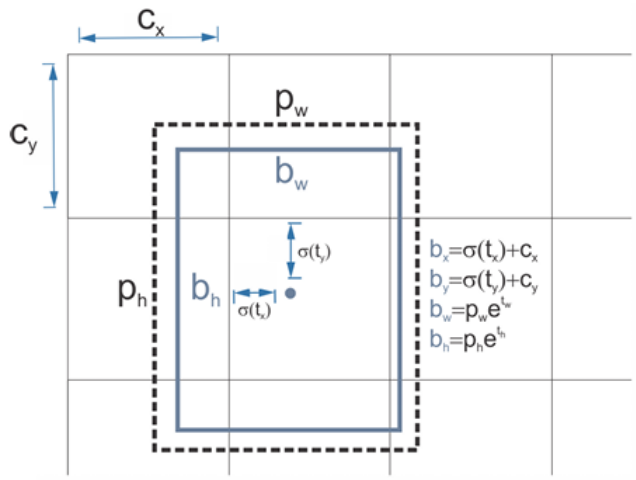

Figure 4.8: yolo v2 anchor boxes

5. Dimension clusters - pick good anchor boxes using k-means clustering on the training bounding boxes - improves accuracy of bounding boxes
6. Direct Location Prediction
7. Finer-grained features
    1. removed pooling layer - get feature 13 x 13 feature map for 416 x 416 images
    2. passthrough layer - 26 x 26 x 512 feature map -> stack adjacent features into different channels
8. Multi-scale training - train on different input sizes to make model robust to different input types

4.2.4.2.1 v2 Architecture

- backbone architecture -> Darknet-19
    - 19 conv layers
    - 5 max pool layers
        - non-linear operation - uses OT to perform efficiently
    - use 1×1 conv between 3×3 to reduce parameters
    - batch normalization to help convergence
    - object classification head (replaces last 4 conv layers of YOLOv1)
        - 1 conv layer (1000 filters)
        - GAP layer
        - Softmax classifier

4.2.4.3 YOLOv3

MS COCO AP: 36.2% AP(50): 60.6%

The code used to run YOLOv3 in `Torch` is provided at this repository

4.2.4.3.1 YOLOv3 Architecture

YOLOv3 makes use of a larger network architecture (backbone) called Darknet-53.

- replaces all max-pooling layers with **strided convolutions** and added residual connections (what are residual connections?) - see Chapter 3 for more information on this primitive.

The darknet architecture is presented here as well (visually):

Num	Type	Filters	Size/Stride	Output
1	Conv/BN	32	3×3 / 1	$416 \times 416 \times 32$
2	Max Pool		2×2 / 2	$208 \times 208 \times 32$
3	Conv/BN	64	3×3 / 1	$208 \times 208 \times 64$
4	Max Pool		2×2 / 2	$104 \times 104 \times 64$
5	Conv/BN	128	3×3 / 1	$104 \times 104 \times 128$
6	Conv/BN	64	1×1 / 1	$104 \times 104 \times 64$
7	Conv/BN	128	3×3 / 1	$104 \times 104 \times 128$
8	Max Pool		2×2 / 2	$52 \times 52 \times 128$
9	Conv/BN	256	3×3 / 1	$52 \times 52 \times 256$
10	Conv/BN	128	1×1 / 1	$52 \times 52 \times 128$
11	Conv/BN	256	3×3 / 1	$52 \times 52 \times 256$
12	Max Pool		2×2 / 2	$52 \times 52 \times 256$
13	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
14	Conv/BN	256	1×1 / 1	$26 \times 26 \times 256$
15	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
16	Conv/BN	256	1×1 / 1	$26 \times 26 \times 256$
17	Conv/BN	512	3×3 / 1	$26 \times 26 \times 512$
18	Max Pool		2×2 / 2	$13 \times 13 \times 512$
19	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
20	Conv/BN	512	1×1 / 1	$13 \times 13 \times 512$
21	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
22	Conv/BN	512	1×1 / 1	$13 \times 13 \times 512$
23	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
24	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
25	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
26	Reorg layer 17			$13 \times 13 \times 2048$
27	Concat 25 and 26			$13 \times 13 \times 3072$
28	Conv/BN	1024	3×3 / 1	$13 \times 13 \times 1024$
29	Conv	125	1×1 / 1	$13 \times 13 \times 125$

Figure 4.9: yolo v2 architecture

Layer	Filters	size	Repeat		Output size
Image					416 × 416
Conv	32	3 × 3/1	1		416 × 416
Conv	64	3 × 3/2	1		208 × 208
Conv	32	1 × 1/1	Conv		208 × 208
Conv	64	3 × 3/1	Conv	×1	208 × 208
Residual			Residual		208 × 208
Conv	128	3 × 3/2	1		104 × 104
Conv	64	1 × 1/1	Conv		104 × 104
Conv	128	3 × 3/1	Conv	×2	104 × 104
Residual			Residual		104 × 104
Conv	256	3 × 3/2	1		52 × 52
Conv	128	1 × 1/1	Conv		52 × 52
Conv	256	3 × 3/1	Conv	×8	52 × 52
Residual			Residual		52 × 52
Conv	512	3 × 3/2	1		26 × 26
Conv	256	1 × 1/1	Conv		26 × 26
Conv	512	3 × 3/1	Conv	×8	26 × 26
Residual			Residual		26 × 26
Conv	1024	3 × 3/2	1		13 × 13
Conv	512	1 × 1/1	Conv		13 × 13
Conv	1024	3 × 3/1	Conv	×4	13 × 13
Residual			Residual		13 × 13

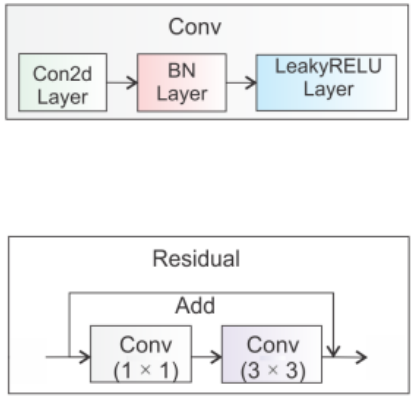

Figure 4.10: yolo v3 architecture

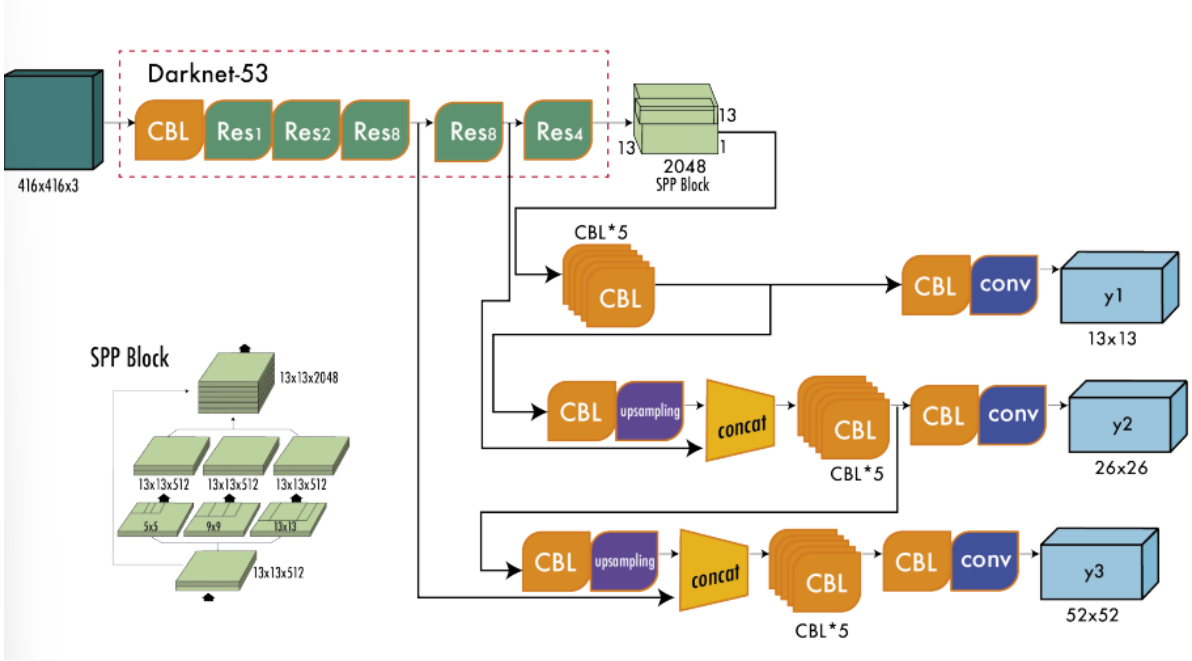

Figure 4.11: darknet 53 backbone

4.2.4.3.2 Multi-Scale predictions

- enables multi-scale predictions (predictions at multiple grid sizes)
- this helps to obtain finer detailed boxes (improves prediction of smaller boxes)
- YOLOv3 generates three separate outputs:
  - **y1**: 13×13 grid defines the output
  - **y2**: concatenating output after $(Res \times 4)$ with output of $(Res \times 8)$ - upsampling occurs from **y1** since the feature maps are of different sizes (13×13) and (26×26)
  - **y3**: upsample **y2** output to match 52×52 feature maps

4.2.4.3.3 Backbone, Neck, and Head

After release of YOLOv3, object detectors began to be described in terms of the backbone, neck, and head

Backbone

- Extracts useful features from the input image.
- A convolutional nerual network trained on large-scale image classifications task (ImageNet)
- captures hierarchical features at different scales

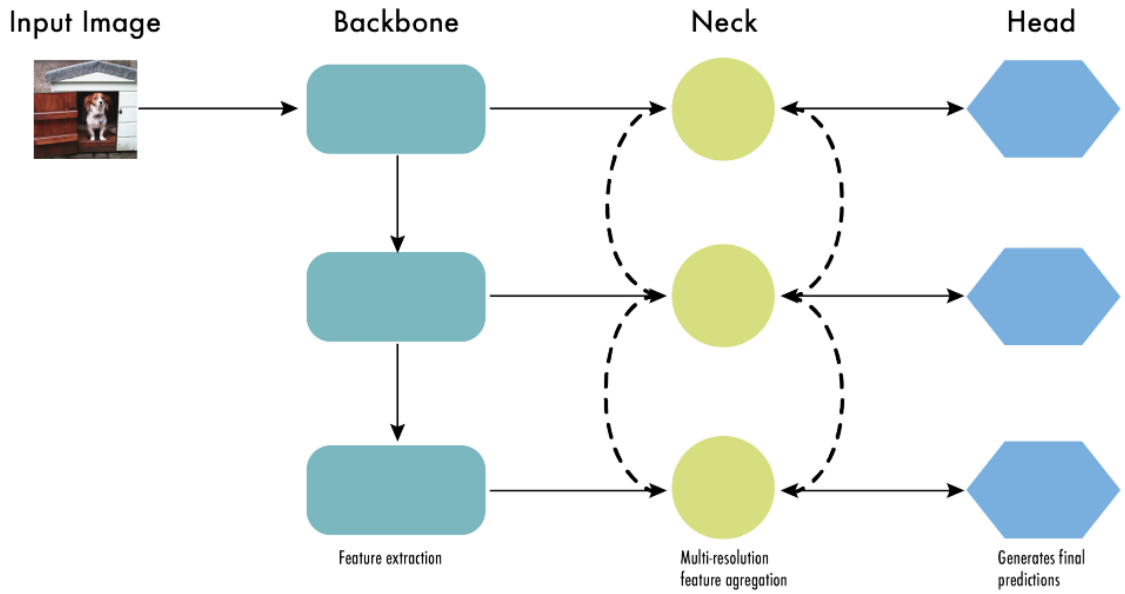

Figure 4.12: modern object detection architecture

  - low-level features - earlier layers
  - high-level features - deeper layers

Neck

- aggregates / refines features extracted by backbone
  - enhance spatial / semantic information across different scales
  - includes conv layers
  - includes feature pyramid networks

Head

- makes predictions based on features provided by backbone and neck
- consists of task-specific subnetworks to perform *classification*, *localization*, *localization*, *instance segmentation* and *pose estimation*
- non-maximum supression filters out overlapping predictions (retains only most confident detections)

4.2.4.4 YOLOv4

MS COCO AP: 43.5% AP(50): 65.7%

The philosophy of YOLOv4 approaches optimization of the model into two categories: *bag of freebies* and *bag of specials*

Bag of Freebies:

- increase training time / cost
- do not affect inference time
- examples include *data augmentation*

Bag of Specials:

- increase inference time / cost
- improve accuracy of the model (MaP)
- examples include
  - enlarging receptive field ???
  - combining features ???
  - post-processing ???

4.2.4.4.1 Model Improvements

- **Enhanced network architecture with Bag-of-Specials**
  - backbone: **Darknet-53** + Cross-stage Partial Connections (CSPNet) ?? + Mish Activation Function ??
    - *tested several backbone architectures to choose best option*
    - CSP reduces computation while maintaining accuracy
  - neck: Spatial Pyramid Pooling (SPP)?? + **Multi-scale predictions** + modified path aggregation network PANet + modified spatial attention module (SAM)
    - SPP increases receptive field without affecting inference speed
  - detection head: **Same anchors as YOLOv3**
- **Advanced training with Bag-of-Freebies**
  - regular augmentation
    - random brightness, contrast, scaling, cropping, flipping, rotation
  - special
    - **mosaic augmentation**:
      - combines four images into a single one
      - reduces need for large mini-batches for batch normalization
    - **DropBlock** regularization (instead of *Dropout*)
    - **CIoU** loss and Cross mini-batch nomralization **CmBN** for collecting statistics from the entire batch instead of just mini-batches

- these changes improve the **detector**

- **Self-adversarial Training**
  - improves model robustness to perturbations
- **Hyperparameter optimization with Genetic Algorithms**
  - genetic algs used on first 10% of periods
  - cosine annealing scheduler to alter learning rate during training

4.2.4.4.2 YOLOv4 Architecture

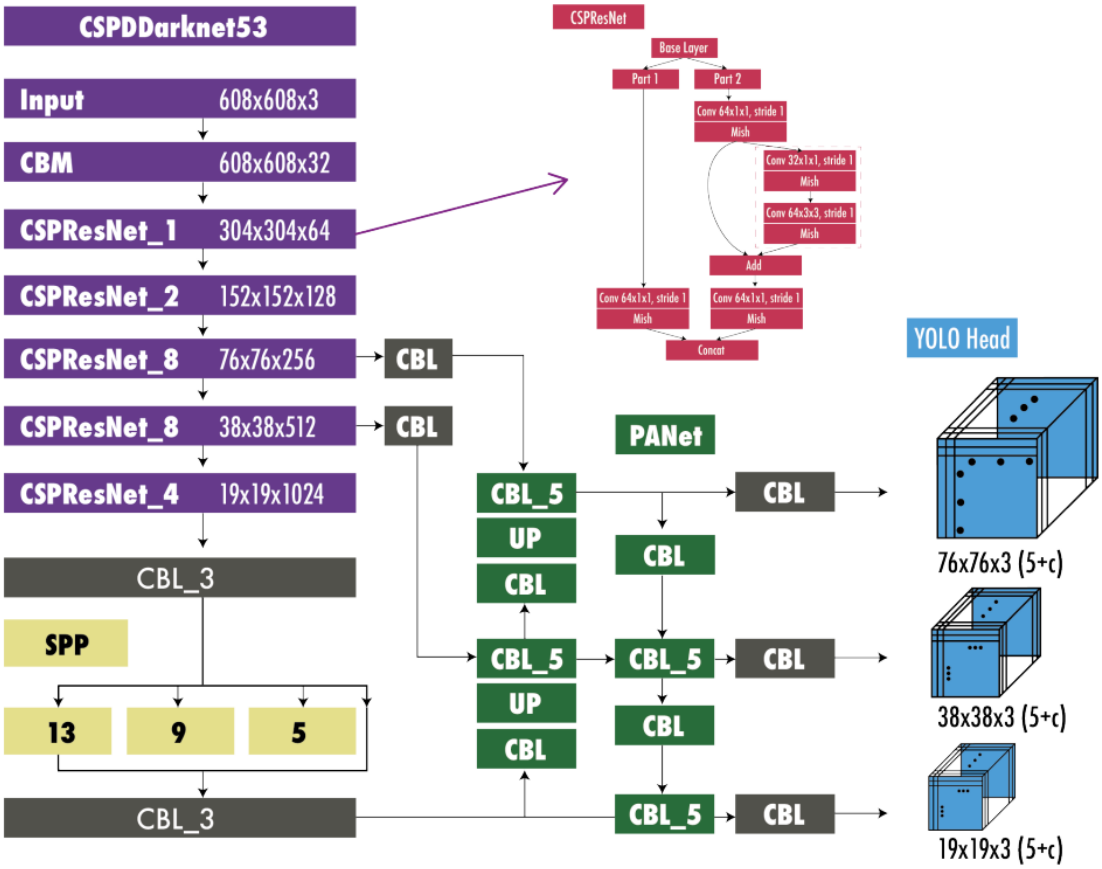

Figure 4.13: yolo v4 architecture

4.2.4.5 YOLOv5

YOLOv5x MS COCO AP: 50.7% AP(50): X%

No paper exists for YOLOv5 but there are several key advantages to using it

- It is developed in PyTorch instead of Darknet
- is open source and actively maintained by ultralytics
- is easy to use, train, and deploy
- many integrations for labeling, training, and deployment (mobile)

There are several scaled versions of this model:

- **YOLOv5n** (nano)
- **YOLOv5s** (small)
- **YOLOv5m** (medium)
- **YOLOv5l** (large)
- **YOLOv5x** (extra large)

This is useful for V2X privacy preserving applications because we will want to make use of smaller networks than might otherwise be used and compare their performance (efficiency) against larger networks.

This helps to answer the question of whether the accuracy tradeoff is worth the speedup.

4.2.4.5.1 Results

YOLOv5x achieves the following results

- AP 50.7% on COCO [batch size = 32, 640 pixels]
- AP 55.8% on COCO [1536 pixels]

4.2.4.6 Scaled-YOLOv4

YOLOv4-large MS COCO AP: 55.6%

Utilized scaling techniques (making larger to increase accuracy at the expense of speed, and scaling down increases speed at the expense of accuracy). Scaled-down models require less compute power and can run on embedded systems

Like YOLOv5, was developed in **PyTorch** instead of **Darknet**

4.2.4.6.1 Results

- YOLOv4-large - MS COCO AP: 56%
- YOLOv4-tiny - MS COCO AP: 22%

4.2.4.7 YOLOR

MS COCO AP: 55.4% AP(50): 73.3%

YOLOR stands for "you only learn one representation" and is novel for introducing multi-task learnin approach which creates a single model for multiple tasks (classification, detection, pose estimation) by learning a general representation and using sub-networks to create task-specific representations!

4.2.4.7.1 Results

- YOLOR - MS COCO AP: 55.4% AP(50): 73.3%

4.2.4.8 YOLOX

MS COCO AP: 50.1%

Was designed off the back of the Ultralytics YOLOv3 (see Section 4.2.4.3) in PyTorch.

4.2.4.8.1 Model Changes and Improvements

- Anchor Free - simplified training and decoding process
- Multi Positives - center sampling (assigned center 3×3 area as positives) to account for imbalances produced by lack of anchors
- Decoupled Head - classification confidence and localization accuracy separated into two heads (connecting them leads to some misalignment)
  - **sped up model convergence** and **improved accuracy**
- Advanced Label Assignment - ambiguities were associated with ground truth bounding boxes (box overlap) - this is addressed with assigning procedure as Optimal Transport Problem . The authors develop a simplified version called simOTA
- Strong augmentations - MixUP and Mosaic augmentations were used

4.2.4.8.2 Architecture

4.2.4.9 YOLOv6

MS COCO AP: 52.5% AP(50): 70%

Adopted an anchor-free detector and provided a series of models at different scales for nuanced applications.

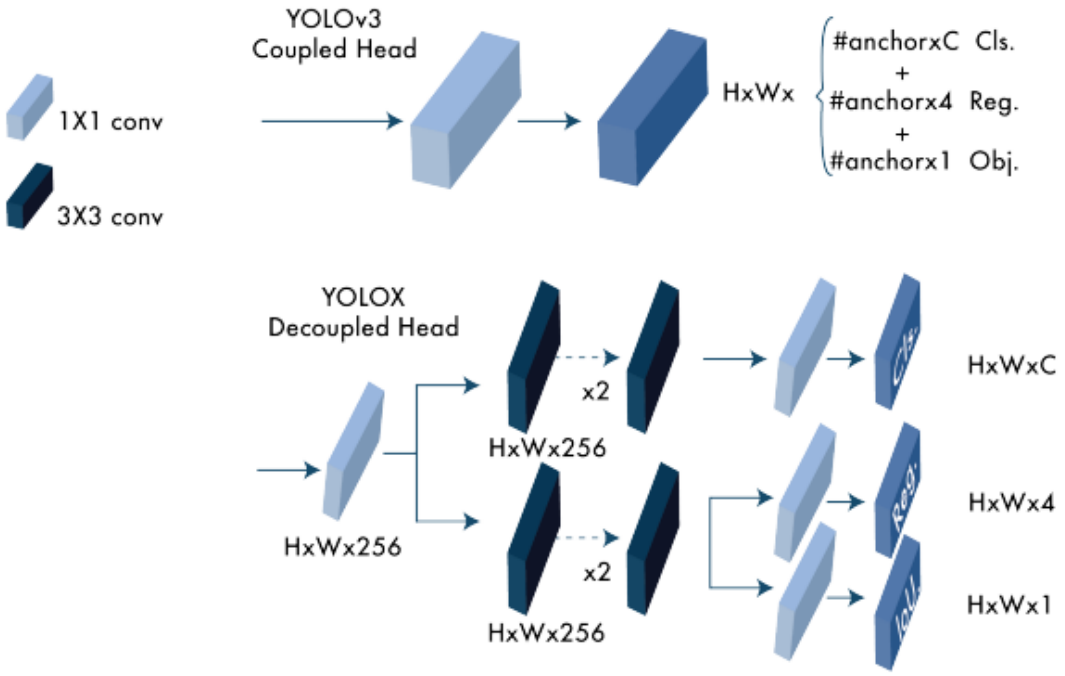

Figure 4.14: yolox architecture

4.2.4.9.1 Model Changes and Improvements

- New architecture backbone based on RepVGG . higher parallelism and use of neck based on RepBlocks or CSPStackRep Blocks and developed an efficient decoupld head
- Label Assignment with TOOD
- New classification and regression losses using VariFocal loss and SIoU / GioU regression loss
- self-distillation for regression and classification tasks
- quantization scheme for detection with RepOptimizer channel-wise distillation for a faster detector

4.2.4.10 YOLOv7

MS COCO AP: 55.9% AP(50): 73.5% - input 1280 pixels

Compared to YOLOv4 and YOLOR, YOLOv7 achieves a significant reduction in parameters used, and improves average precision by a meaningful increase in average precision as well.

4.2.4.10.1 Model Changes and Improvements

Architectural Changes

- Extended efficient layer aggregation network - ELAN allows for more efficient training and convergence
- Model Scaling for concatenation-based models - maintain optimal structure of the model by scaling thedepth and width of the block with the same factor

Bag-of-Freebies

- Planned re-parameterized convolution - Architecture is inspired by RepConv . Identity connection outperforms residual in Resnet (Chapter 3) and concatenation in DenseNet
- Coarse label assignment for auxiliary head (training) and fine label assignment for the lead head (final out)
- Batch normalization in conv-bn-activation - integrates mean and variance of batch normalization into weight and bias of convolutional layer at inference stage (batch norm folding)
- Implicit knowledge (see Section 4.2.4.10)
- Exponential moving average as final inference model

4.2.4.11 DAMO-YOLO

MS COCO AP: 50.0%

Was inspired by a series of technologies relevant at the time and provided tiny/small/medium scaled model variants

- Neural architecture search (this was also used in Delphi) - MAE NAS -
- a large neck
- a small head
- aligned OTA label assignment - uses focal loss for classification cost and IoU of prediction / ground truth box as *soft label*. Enables selection of *aligned samples* for targets
- knowledge distillation
  - (1) teacher guiding student (stage 1)
  - (2) student fine-tuning (stage 2)
  - enhancements in distillation approach
    - align module - adapts student features to same resolution as teacher's
    - channel-wise dynamic temperature - normalizes teacher and student features

4.2.4.12 YOLOv8

YOLOv8x MS COCO AP: 53.9% (640 pixels)

A version of YOLO released by ultralytics which is anchor free and uses mosaic augmentation for training up to the last ten epochs. It provides five scaled versions

1. YOLOv8n (nano)
2. YOLOv8s (small)
3. YOLOv8m (medium)
4. YOLOv8l (large)
5. YOLOv8x (extra large)

4.2.5 PP-YOLO Models

The PP-YOLO models were developed in parallel with the standard YOLO variants. These models were based on YOLOv3 but were developed in the PaddlePaddle deep learning platform instead. The goal of their work was to show how an object detector should be constructed step-by-step, not to provide any novel functionality or a new approach.

4.2.5.1 PP-YOLO

MS COCO AP:45.9% AP(50): 65.2%

4.2.5.1.1 Divergence from YOLOv3

- ResNet50-vd backbone + deformable convolutions replace DarkNet-53 (Section 4.2.4.3) - achieves higher classification accuracy on Imagenet
- Larger batch size - improves training stability
- Maintained moving averages for trained parameters
- DropBlock applied on FPN
- IoU loss added with L1-Loss for bounding box regression
- IoU prediction branch for measuring localization accuracy (and optimization)
- Grid Sensitive Approach like YOLOv4 (Section 4.2.4.4) - improves bounding box center prediction at grid boundary
- Matrix NMS (parallelized NMS for faster computation)
- CoordConv - 1×1 convolution of the FPN. Allows for learning translational invariance
- Spatial Pyramid Pooling increases receptive field of the backbone

4.2.5.1.2 PP-YOLO Augmentations and Preprocessing

- Mixup Training - weights $\sim Beta(\alpha = 1.5, \beta = 1.5)$
- Random color distortion
- Random expand
- Random crop and random flip with probability 0.5
- RGB channel z-score normalization $\mu = [0.485, 0.456, 0.406]$ and $\sigma = [0.229, 0.224, 0.225]$

4.2.5.2 PP-YOLOv2

MS COCO AP: 49.5%

4.2.5.2.1 Changes and Improvements

- Backbone changed from ResNet50 to ResNet101
- Path aggregation network instead of FPN
- Mish Activation function applied to the detection neck
- Larger input sizes - increases performance on small objects
- Modified IoU aware branch - soft label format instead of soft weight format

4.2.5.3 PP-YOLOE

MS COCO AP: 51.4% (78.1 FPS NVIDIA V100)

4.2.5.3.1 Changes and Improvements

- Anchor Free (following trends of other yolo models)
- New backbone and neck - modified neck wtih RepResBlocks (combine dense and residual connections)
- task alignment learning (see Section 4.2.4.8)
- efficient task-aligned head (ET-head) - single head based on TOOD instead of splitting the classification / detection heads like with YOLOX (Section 4.2.4.8)
- varifocal (VFL) and distribution focal loss (DFL)
  - VFL weights positive samples using a target score which places a greater importance on high-quality samples during training
  - DFL extends Focal Loss from discrete to continuous labels - better for representations which combine qualtiy estimation and class prediction - this allows for better depiction of flexible distributions in real data (eliminates risk of inconsistency)

The authors provide several scaled modesl

- PP-YOLOE-s (small)
- PP-YOLOE-m (medium)
- PP-YOLOE-l (large)
- PP-YOLOE-x (extra large)

4.2.6 Summary of YOLO

Table 4: Summary of YOLO architectures. The metric reported for YOLO and YOLOv2 were on VOC2007, while the rest are reported on COCO2017

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*63.4*
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	Modified CSP v7	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	Modified CSP v5	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	RepConvN	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLO v8	53.9

Figure 4.15: summary of yolo architectures and results

Part III

V2X Applications

This portion of the text deals specifically with discussions of V2X applications and papers which have tackled these sorts of the problems in the past.

5 Red Light Violation Detection

5.1 Motivation

Secure Red Light Violation detection is an important application of secure machine learning protocols. Oftentimes, these systems will require the use of image segmentation and object recognition protocols. The images used for this task expose often-times sensitive data about individual users.

In the practical setting of interest, this means exposing license-plate information, associated vehicles, and location information available from the images themselves.

5.2 V2I Algorithms for RLR Detection

> The authors of this paper have constructed V2I mechanisms for red light running (RLR) detection, wrong way entry (WWE), and an array of other import tasks in the context of V2X. See the citation Dokur and Katkoori (2022)

5.2.1 Red Light Violation Detection Algorithm

The proposed system utilizes the following logic to detect whether a car will violate a red light. A car which is approaching an intersection is connected to road-side units (RSUs) which are installed at traffic lights in an intersection.

Each light is said to be located at points $B(x_2, y_2, z_2)$, $C(x_3, y_3, z_3)$, $D(x_4, y_4, z_4)$ and $E(x_5, y_5, z_5)$ respectively.

Unlike image-based systems, this system assumes V2I communication between the traffic lights and the vehicle in question. This means that the traffic state does not need to be determined by an image classifier. Rather, we already have this information by default.

5.3 Thao et.al on Traffic Violation Detection

The paper proposed by L. Thao (2022) introduces a mechanism for detecting red light violations automatically. There paper is titled: *Automatic Traffic Red-Light Violation Detection Using AI*

5.3.1 Problem Setting

The reason AI technologies (image classification and detection) systems are better suited than standard sensor technologies is that they are able to operate more consistently, even when the number of vehicles in the setting increases dramatically.

5.3.2 System Design and Solution Approach

Separate the task into three parts:

1. vehicle violation detection
2. red signal change monitoring
3. vehicle recognition

Vehicle Violation Detection

The YOLOv5s pretrained model (COCO dataset) is used for detecting violating vehicles. After detecting violation, following frames are used to try and determine the license plate (identfy vehicle). See Section 4.2.4.1 for more information on the YOLO object detection model.

Below, a picture of the overall system flow is presented:

- **vehicle tracking** - performed every 5 frames
  - if IOU (intersection over union) of bounding box is close to one from a previous frame, then the car is assumed to be the same one from that frame.
- **violation line detection**
  - image processing is used to determine traffic lines
  - boundary lines are drawn onto frames captured by the camera later
- **traffic state detection**
  - color filters and image processing used to detect changes in the state of the traffic light

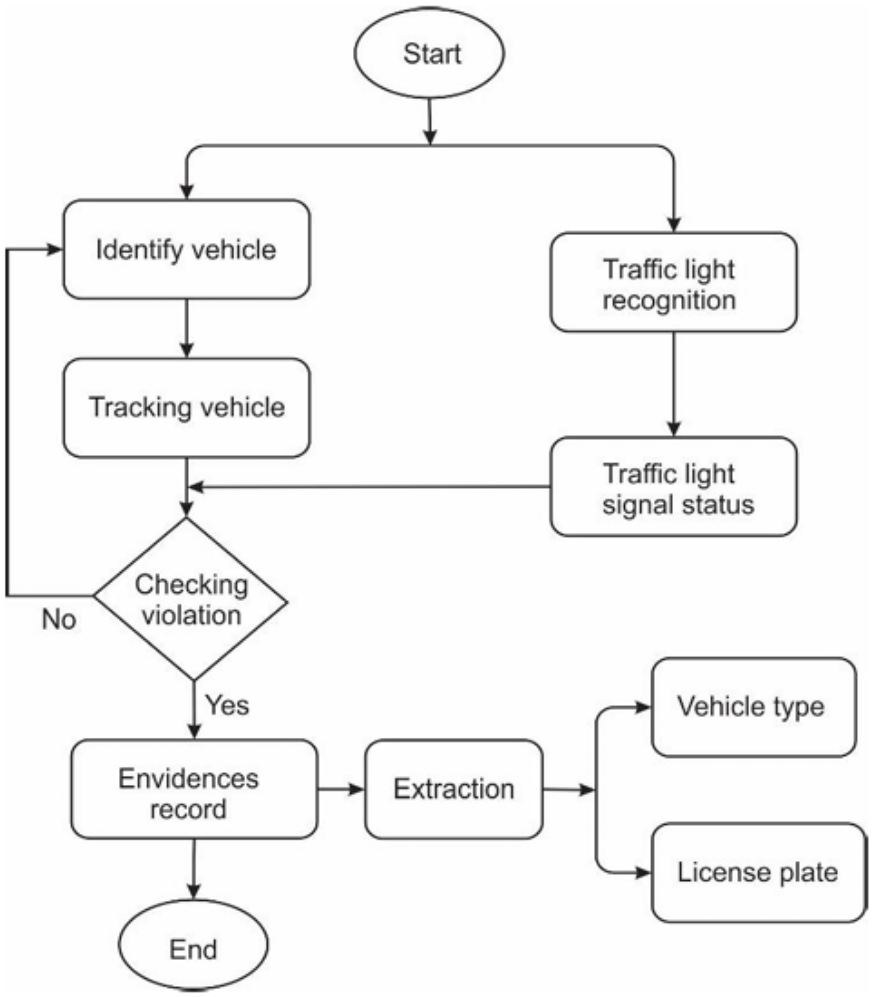

Figure 2. Algorithm of violation detection

Figure 5.1: system flow

5.3.3 Primary Contributions

1. Implementation of modified *YOLOv5s* model
    1. used parameter changes from original model
    2. achieved following accuracy results:
        1. 82% - vehicle identification
        2. 90% - traffic signal status change
        3. 86% - violation detection
2. Best Performing Architecture given below (v3 / v4)

Figure 9. YOLOV5S_v2 remove y3 block

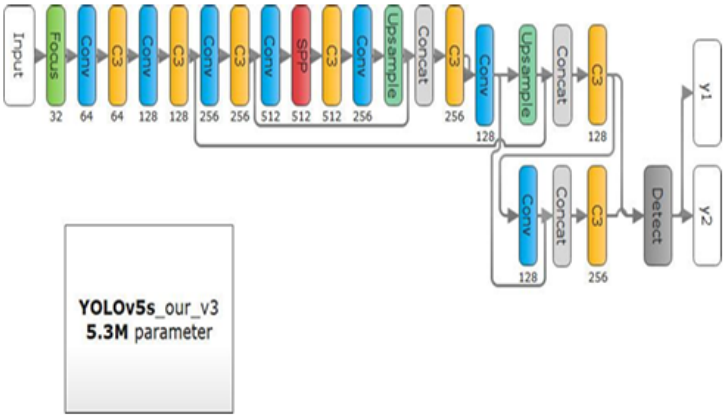

Figure 10. YOLOV5S_v3 reduce ½ filter in Conv, remove y3 block

Figure 5.2: modified Yolo architecture

According to a labmate, it may only be necessary to fine tune the YOLO model on 10 epochs. It may not be necessary to do any more than that.

5.4 Goma et.al on RLR Detection with SSD (Single Shot Detector)

Work by J. Goma (2020) demonstrates the ability to detect red-light-running and over-speeding with a high level of accuracy. Specifically, they achieve **100%** accuracy on red light running detection and **92.1%** accuracy for over-speeding violations. They accomplish these results using a CNN applied to an SSD (single deep neural network)

5.4.1 Methods and System Design

6 Traffic Flow Forecasting

6.1 Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting

In this paper, S. Guo (2019) proposed a method for traffic flow prediction which utilized an **attention based spatial-temporal graph convolutional network**. They aimed to model several time-dependencies: (1) recent, (2) daily-periodic, and (3) weekly-periodic dependencies. The *attention mechanism* captures spatial-temporal patterns in the traffic data and the *spatial-temporal convolution* is used to capture spatial patterns while *standard convolutions* describe temporal features.

6.1.1 Core Contributions of ASTGCN

Difficulties of the traffic forecasting problem

1. it is difficult to handle unstable and nonlinear data
2. prediction performance of models require extensive feature engineering
    1. domain expertise is necessary
3. cnn - spatial feature extraction from grid-based data, gcn - describe spatial correlation of grid based data
    1. fails to simultaneously model spatial temporal features and dynamic correlations of traffic data

Addressing these issues:

1. develop a *spatial-temporal attention mechanism*
    1. learns dynamic spatial-temporal correlations of traffic data
    2. temporal attention is applied to capture dynamic temporal correlations for different times
2. Design of *spatial-temporal convolution module*
    1. has graph convolution for modeling graph structure
    2. has convolution in temporal dimension (kind of like 3-d convolution)

7 Summary

In summary, this book has no content whatsoever.

References

Beaver, D. 1991. "Efficient Multiparty Protocols Using Circuit Randomization." *Advances in Cryptology*. https://link.springer.com/chapter/10.1007/3-540-46766-1_34.

Dokur, O., and S. Katkoori. 2022. "Vehicle-to-Infrastructure Based Algorithms for Traffic Light Detection, Red Light Violation, and Wrong-Way Entry Applications." *IEEE International Symposium on Smart Electronic Systems*.

J. Goma, R. Bautista, M. Eviota. 2020. "Detecting Red-Light Runners (RLR) and Speeding Violtion Through Video Capture." *IEEE 7th International Conference on Industrial Engineering and Applications*.

J. Terven, D. Cordova-Esparaza. 2023. "A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond." *ACM Computing Surveys*. https://arxiv.org/pdf/2304.00501v1.pdf.

L. Thao, N. Anh, D. Cuong. 2022. "Automatic Traffic Red-Light Violation Detection Using AI." *International Information and Engineering Technology Association*.

S. Guo, N. Feng, Y. Lin. 2019. "Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting." *The Thirty-Third AAAI Conference on Artificial Intelligence*.

W. Liu, D. Erhan, D. Anguelov. 2016. "SSD: Single Shot MultiBox Detector." *ECCV*.

Wong, W. 2021. "What Is Residual Connection?" https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55.