# MLHD: Machine Learning for Human Detection — Building an Object Detector from Scratch

Daveed Vodonenko Lucas Portela Darius Vulturu

## Motivation & Problem Statement

Detecting objects in images is a fundamental task in computer vision, powering applications from self-driving cars to medical imaging. However, most state-of-the-art detectors are trained on curated, high-resolution datasets and often fail under real-world CCTV conditions: low light, occlusions, motion blur, and unusual camera angles. Our project, **MLHD** (Machine Learning for Human Detection), addresses this gap by focusing specifically on the **detection of humans in noisy surveillance video**. This is a critical first step toward spatio-temporal action recognition, which underpins safety analytics in industrial and urban environments. Our **hypothesis** is that a compact convolutional neural network (which learns visual patterns from images using filters), trained end-to-end on real CCTV frames, can reliably localize people despite these challenges. This provides a strong foundation for downstream tasks that could save lives, such as near-miss detection and accident prevention.

## Objectives

We have four objectives. The main goal is to train a human-detection model optimized for real-world CCTV video. On the educational side, we aim to deepen our understanding of convolutional neural networks by manually implementing training loops using PyTorch tensors and low-level operations. From an applied perspective, we seek to achieve reliable Intersection-over-Union (IoU) scores for human detection on CCTV validation footage, ensuring that predicted bounding boxes closely overlap with the ground-truth annotations. Finally, on the research side, we plan to compare the performance of our minimalist detector to published baselines such as YOLOv4 [2] in domain-shifted scenarios.

## Implementation Overview

We base our approach on the YOLO framework, **You Only Look Once** [1,2], a real-time object detection system that divides an image into a grid and predicts bounding boxes and class probabilities for each cell in a single forward pass. YOLO is fast, end-to-end trainable, and particularly suitable for video surveillance tasks. In our project, a small CNN backbone extracts features, and a detection head predicts objectness and bounding boxes per cell on an $S \times S$ grid. Training uses a composite loss function: binary cross-entropy (for objectness) [3] and coordinate regression loss (for bounding box localization). Full equations and layer specifications are detailed in Annex A: Technical Details (see page 3).

## Data & Work Plan

We will collect CCTV footage (obtained from one of our team members´ independent projects) from multiple cameras, bootstrap annotations with an existing detector [1], and manually clean samples to ensure quality. **Phase 1:** Data prep, grid encoding, baseline model (Weeks 1–3). **Phase 2:** Training and hyperparameter search (Weeks 4–6). **Phase 3:** Evaluation, visualization, and final report (Weeks 7–8).

## Impact

This project produces both an educational artifact, a from-scratch PyTorch training pipeline, and a practical human-detection model, which can be extended to temporal action recognition to improve workplace and urban safety.

## References


1. Redmon, J. et al., "You Only Look Once: Unified, Real-Time Object Detection," *CVPR 2016*.
2. Bochkovskiy, A. et al., "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv:2004.10934*.
3. Lin, T.Y. et al., "Focal Loss for Dense Object Detection," *ICCV 2017*.

# Annex A: Technical Details

## 1. Image Representation

We begin with an input image of width $W$ pixels and height $H$ pixels. An image can be represented as a 3D tensor

$$I \in \mathbb{R}^{H \times W \times 3},$$

where the last dimension corresponds to the RGB color channels.

## 2. Grid Division

To detect objects, we divide the image into an $S \times S$ grid. Each grid cell is responsible for predicting whether an object's center falls inside it.

- $S$ = number of cells along one dimension (e.g., $S = 26$ for a $416 \times 416$ input).
- Each cell therefore covers $\frac{W}{S} \times \frac{H}{S}$ pixels.

## 3. Model Architecture

The network has two main components:

- **Backbone:** A sequence of convolutional layers with increasing channel depth (16, 32, 64, 128, 256, 512) and stride-2 downsampling. For an input image of size $416 \times 416$, this produces a feature map of size $(512, 26, 26)$. The backbone's role is to extract spatial features from the raw pixels.
- **Detection Head:** A $1 \times 1$ convolution maps each cell's 512-dimensional feature vector to five outputs:

  $$(\hat{t}_x, \hat{t}_y, \hat{t}_w, \hat{t}_h, \hat{t}_{obj}).$$

  Here:

  - $\hat{t}_x, \hat{t}_y$ = predicted offset of the object center inside the grid cell,
  - $\hat{t}_w, \hat{t}_h$ = predicted normalized width and height,
  - $\hat{t}_{obj}$ = predicted objectness score (probability an object is present).

  **Activation Functions:** All outputs pass through a sigmoid activation, constraining values to $[0, 1]$.

- Offsets $(\hat{t}_x, \hat{t}_y)$ must remain within a cell.
- Width and height $(\hat{t}_w, \hat{t}_h)$ are expressed as fractions of the image size.
- Objectness $\hat{t}_{obj}$ is a probability.

## 4. Bounding Box Definition

A ground-truth bounding box is defined by its pixel coordinates:

$$(x_1, y_1) \quad \text{(top-left)}, \qquad (x_2, y_2) \quad \text{(bottom-right)}.$$

We then compute normalized parameters:

$$c_x = \frac{x_1 + x_2}{2W}, \quad c_y = \frac{y_1 + y_2}{2H}, \quad w = \frac{x_2 - x_1}{W}, \quad h = \frac{y_2 - y_1}{H}.$$

Here $(c_x, c_y)$ is the box center relative to the image size, and $(w, h)$ are normalized width and height.

## 5. Assigning to Grid Cells

We determine the grid cell responsible for predicting the object:

$$i = \lfloor c_x S \rfloor, \quad j = \lfloor c_y S \rfloor,$$

where $i$ is the column index and $j$ is the row index.

## 6. Target Tensor Construction

The training target is a tensor $Y \in \mathbb{R}^{S\times S\times 5}$. Each cell $(j, i)$ stores:

$$Y[j, i] = (t_x^*, t_y^*, t_w^*, t_h^*, t_{obj}^*),$$

where

$$t_x^* = c_x S - i, \quad t_y^* = c_y S - j, \quad t_w^* = w, \quad t_h^* = h, \quad t_{obj}^* = 1.$$

All other cells store zeros.

**Interpretation:**

- $(t_x^*, t_y^*)$ = offset of the object center inside the cell, in $[0, 1]$,
- $(t_w^*, t_h^*)$ = normalized width and height,
- $t_{obj}^* = 1$ if the cell contains an object center, else 0.

## 7. Loss Function

The total training loss is a weighted sum of localization and objectness losses:

$$L = \lambda_{coord} L_{coord} + L_{obj}.$$

**(a) Localization loss:** Applied only to cells with $t_{obj}^* = 1$:

$$L_{coord} = (\hat{t}_x - t_x^*)^2 + (\hat{t}_y - t_y^*)^2 + (\hat{t}_w^2 - (t_w^*)^2)^2 + (\hat{t}_h^2 - (t_h^*)^2)^2.$$

**(b) Objectness loss:** For each cell $(i, j)$:

$$L_{obj} = \sum_{i,j} \Big[ M_{ij}^{pos} \,\mathrm{BCE}(\hat{t}_{ij}^{obj}, 1) + \lambda_{noobj} M_{ij}^{neg} \,\mathrm{BCE}(\hat{t}_{ij}^{obj}, 0) \Big],$$

where:

- $M_{ij}^{pos} = 1$ if an object center lies in cell $(i, j)$, else 0,
- $M_{ij}^{neg} = 1 - M_{ij}^{pos}$,
- BCE = binary cross-entropy,
- $\lambda_{coord}, \lambda_{noobj}$ are hyperparameters controlling the relative weight of localization and negative objectness terms. Intuitively:
  - $\lambda_{coord}$ increases the emphasis on precise box regression. A higher value is useful if the model predicts object locations but with poor accuracy.
  - $\lambda_{noobj}$ reduces the impact of the many background (no-object) cells. Lowering it prevents the model from being overwhelmed by negatives, while raising it makes the detector stricter about false positives.

## PyTorch Implementation Notes

We use `torch.nn.functional.conv2d`, `F.interpolate` for bilinear upsampling, and custom-written loss functions leveraging `torch.tensor` broadcasting. Backprop and parameter updates are handled by `torch.autograd` and `torch.optim.SGD`.