CMP 7200: Research Project

Birmingham City University

Faculty of Computing Engineering and Built Environment

Floorplan Analysis with Object Detection and Active Learning

Author:

Marios Kyriacou

Supervisor:

Mohammed M. Abdelsamea

September 2022

Acknowledgements

I want to thank my supervisor, Professor Mohammed Abdelsamea, who provided valuable guidance in developing the research topic, and his leadership throughout the process. Finally, I would like to acknowledge my family for their support throughout this year, and more specifically, I would like to thank my father (KK), my mother (AK), my two cousins (P&I D) and my friend (PK) for their support.

Abstract

Artificial Intelligence (AI) is getting more intelligent as it is exposed to machines for recognition. Image recognition and object detection assist computer vision in seeing the world around it and adequately identifying objects. Object detection (OD) is essential for comprehending floorplans and is a prerequisite for converting them to other formats. However, the procedure of labeling a large amount of data for either classification or object detection tasks is both challenging and costly. By deciding the optimal samples from the unlabeled pool, Active Learning (AL) for object detection attempts to reduce annotation costs. In this project, we specifically train an object detection algorithm known as YOLOv5 to distinguish items including doors, windows, and rooms (or zones) in floorplan images. Since the dataset in this study is created from scratch with the assistance of the LabelImg program, the algorithms' training was challenging. Moreover, the images are clustered based on an uncertainty measure known as Least Confidence (LC), and the algorithm is trained until the unlabeled images pool gets empty. Detecting floorplan objects is simple for humans; however, analyzing floorplans and recognizing things automatically with the least amount of data is a challenging task for a Deep Learning model to handle.

Keywords: Computer Vision, Active Learning, Object Detection, YOLOv5, Floorplans

Contents

List of Figures

1 Introduction

1.1 Floorplans analysis with Object Detection

Artificial intelligence's field of computer vision enables machines to think and analyze visual information similarly to how humans do. The field of computer vision has experienced remarkable development, in recent years, with highly practical applications like face time detection, self-driving cars, security cameras, and many others; computer vision is quickly becoming a reality and has made impressive strides. Image classification, object localization, object detection, and object segmentation comprise the main categories of computer vision (Demush 2019). Object identification in images is considered a typical problem, although it is not as straightforward for a machine as it is for the human brain. Identifying and detecting objects in images is a topic known as "object detection" in computer vision. To solve this problem, several algorithms have been created recently; among them, Faster R-CNN, Yolo, and SSD are often used. Over the years, advances in computer vision have been made in speed and accuracy. Data is growing at the present rate of development. However, a limited amount of labelled data can only suit the challenges. Unfortunately, labeling data can be expensive since it is typically done manually by experts in the field, and object identification models need a lot of labeled data to be trained (McGrath 2022). The objective of active learning is to speed up the collection of new labeled data and to sort the unlabeled samples according to the value that can be predicted from annotating them.

Active learning has been extensively investigated for classification tasks, while the object detection has received less interest in the research domain. In this thesis, we propose an active learning method for object detection; more specifically, this project uses a pool-based approach to correctly label three distinct classes, named "door," "window," and "room," in a 2D floorplan image. This thesis uses the YOLOv5 model to identify the objects in the floorplan image, while the model is trained in only 400 images performing accurate results, however struggling identifying one of the three classes. Once the active learning approach is used, the model is able to automatically label and classify an image. The dataset consists of 500 unlabeled floorplan images, of which 400 were manually labeled and used as the training set of the model, and the remaining 100 images were labeled by the model and clustered based on their confidence score to retrain the model. Figure 1 visualizes a floorplan image labeled by the model automatically.

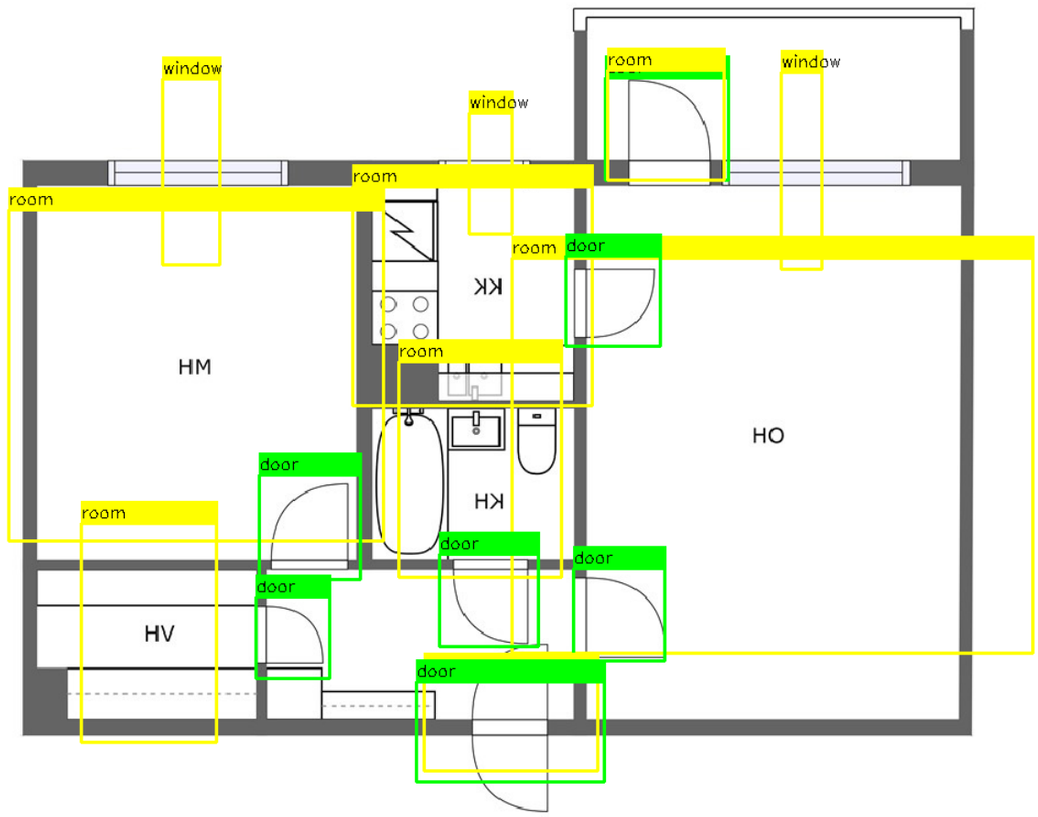

Figure 1: The annotation provided by the model.

1.2 Overview of the structure

The rest of this paper is structured as follows. In Section 2, we give an overview of the related work. Section 3 discusses the theoretical and technical background for the basic knowledge of Convolutional neural networks. The introduction of the YOLO family and the versions of YOLO and their differences in the architecture and the identification process are extensively discuss in Section 3.4. In Section 3.5, we develop the basic methods of Active Learning and the main Uncertainty measures. Section 4 proposes our active learning methodology and distinguishes the proposed algorithm, whilst in Section 5 we present the results of our method. Finally, Section 6 summarizes the proposed methodology and the finding results, and also discusses the future research of our study.

2 Literature Review

Object Detection in Floorplan images is a crucial task to gain a deep understanding and generate accurate descriptions of the architecture of a building. However, it is well known that training and evaluating a Deep Object Detector for floorplan images is a rough mission for computer vision due to the high amount of annotations and the unnatural type of the image. Thus, active learning aims to solve the first of the two problems stated.

Instead of emphasizing the object detection component of active learning, the majority of articles studying this issue concentrate on the classification task. However, most active learning papers for object detection involve either the information given from an image (Brust et al. 2018, Marbinah 2021, Roy et al. 2018) or the information given from a bounding box (Desai & Balasubramanian 2020).

Desai V.S and Balasubramanian N. V (Desai & Balasubramanian 2020) propose a bounding box based querying approach for active learning; over an unlabeled pool of images. Their method selects the most informative subset of a generated bounding box dataset. For example, suppose that the bounding box dataset is created by training an object detection model M on a small subset L_I of their dataset D_I. Once the model is trained, a Bounding box dataset U_B is generated by finding the predicted Bounding boxes with the following function:

$$M(b) = \begin{cases} 1, & prob(b) \leq t_d \\ 0, & otherwise \end{cases} \tag{1}$$

where t_d is a choosing threshold and $prob(b)$ is the probability for the predicted bounding box. When the function $M(b)$ is equal to 1, the images are added to the training set D_I. This procedure is repeated until all the images are labeled. Hence, this method is tested on the Feature Pyramid Network (Lin et al. 2017) and RetinaNet (Li et al. 2020) models which illustrates promising results.

On the other hand, Roy. S, Unmesh. A and Namboodiri V.P (Roy et al. 2018), randomly selected a set of images for annotation and trained in a Single Shot Multibox Detector (SSD) (Zhai et al. 2020) model. Subsequently, they pick a set/batch of images, annotate them and train the model again; this procedure is repeated until the unlabeled pool of images is empty. However, the images selected are based on two popular active learning methods: the White-box and the Black-box. The *white box* methods can clearly describe how a network or model acts and generates predictions since the network or model architecture is known. Note that if the network's or model's features are intelligible and the process is understandable, the network or the model can be defined as a white box (Loyola-Gonzalez 2019). In this project, we will discuss only the *black-box* approach, which is based on the Minmax (mm) method and the Maximum Entropy (ME) method, which includes the two query functions, as shown in the subsequent text.

Assume that there are $b_{i,c}$ bounding boxes for each class c for each image i. *Min-max* (Huang et al. 2010) query function indicates the model's level of confidence for that image as follows

$$\arg\min_{i} \max_{c} \max_{j \in b_{i,c}} prob(j) \tag{2}$$

and the *Entropy* Holub et al. (2008) level of confidence for an image i is given by:

$$\arg\max_{i} \max_{c} - \sum_{j \in b_{i,c}} prob(j) * log(prob(j)) \tag{3}$$

In addition to active learning, a variety of papers uses deep object detection models to identify objects for floorplan images (Barducci & Marinai 2012, Ziran & Marinai 2018, Goyal et al. 2021). Kalervo.A et al. (Kalervo et al. 2019) illustrate an image dataset called CubiCasa5K holding 5000 images that have been categorized into more than 80 types of floorplan objects. Then, this dataset demonstrates an approach that utilizes an enhanced multi-task convolutional neural network (CNN). In addition, Mishra.S et al. (Mishra et al. 2021) examine the implementation of Mask R-CNN with Deformable Convolutional Networks (DCN) (Dai et al. 2017) and Convolutional Neural Networks (CNNs), and they concluded that DCN outperforms CNNs to detect objects in 2D floorplan images. Lastly, Sandelin. F (Sandelin 2019), considers the Mask R-CNN model to detect and generate a segmentation map for both objects for a small floorplan dataset. The results demonstrate that the model is potentially accurate and reliable for floor plan segmentation.

In our case, this project uses YOLOv5 (You Only Look Once) to detect objects including Doors, Windows, and Rooms/Zones in a 2D floorplan image. We have 500 unlabeled images of which the 400 is manually annotated. By using an Active Learning method, known as Uncertainty Sampling, we automatically annotate the remaining 100 images; indicating the 10% of the Dataset. The proposed method clusters the images according to the confidence of each image. This can be achieved with the assistance of an uncertainty measure known as Least Confidence (LC) (Settles & Craven 2008). Thus, we are provided with the confidence of each image and named three clusters "High", "Mid", and "Low". YOLOv5 is then trained using these confidence results and illustrates that even if the model is trained with low-confidence images, the Mean Average Precision (mAP) of the model can be high.

3 Background & Related Work

This section provides our topic's theoretical and technical background, and relevant studies. The section delivers the theoretical background of Convolutional Neural Networks (CNNs) as a subset of Deep Neural Networks (DNNs). Moreover, it illustrates the uses and the theoretical part of Deep Object Detection (DOD) and how CNNs affect and assists DOD. Conclusively, this section demonstrates the theoretical background of Active Learning (AL) in Deep Learning (DL) as a subcluster of Machine Learning (ML) and provides the basic mathematical uncertainty formulas.

3.1 Artificial Neural Networks (ANNs) & Feed Forward Neural Networks (FFNNs)

The Artificial Neural Networks (ANNs) are inspired by the organic brain network translated to the computer and are a combination of Machine Learning, and Deep Learning (Walczak & Cerpa 1999). A Feedforward neural network (see right panel of Figure 2) is ANNs with the information traveling forward in the network since the units' connections do not form a loop (McGonagle. J 2022).

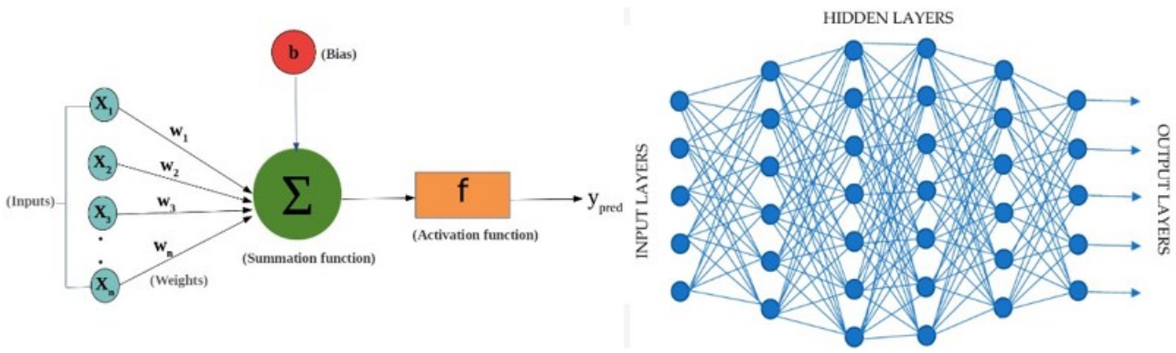

Figure 2: A Deep Neural Network with weight and biases.

As is shown, the left side of Figure 2 illustrates a Deep Neural Network with 5 input nodes, 4 hidden layers (layer in between input layers and output layers (Malik 2019*b*)), and 5 output nodes. As the right side of Figure 2 illustrates, each neuron accepts input (vector) data, either training or the outputs of the last layer. Then, the weights (vector) are applied to the input neurons which demonstrate their effectiveness. Note that, the weights are coefficients of the equation that can be tuned and also reduce the value of an output neuron (Malik 2019*a*). Then bias (constant) is added to the multiplication of the inputs and the weight and the computed value Y is applied to an Activation Function (Ganesh 2020). This can be summarized using the equation 4.

$$ActivationFunction(Y) = ActivationFunction\Big(\sum_{j=1}^{n} input_j * weights_j + bias\Big) \tag{4}$$

Then, the resulting output of the equation 4 is created by utilizing a threshold; an activation function measures how the weighted sum of the input is modified into an output. The most common activation functions used are the Step function, the Sigmoid, the Softmax, and ReLU, that we discuss in the subsequent Sections 3.1.1, 3.1.2, 3.1.3 and 3.1.4, respectively. It is worth

noting that a neural network without an activation function is just a Regression problem (see how the quantity Y is defined in equation 4).

3.1.1 Step Function

To start with, the Step function is one of the most common and simple activation functions. This function produces 1 if the input Y of the activation function is higher than 0, and output the value of 0 otherwise (Serengil 2017).

$$Step(Y) = \begin{cases} 1, & Y > 0, \\ 0, & Y \leq 0 \end{cases} \tag{5}$$

3.1.2 Sigmoid Activation Function

One of the major issues of the step function is that it can not determine accurate information for the optimizer since the output values are either 1 or 0; since it cannot be determined how "near" this step function was to activating or deactivating (SHARMA 2017). Considering this issue for the Step function, we define the Sigmoid Activation Function $\Phi(Y) \in (0, 1)$ as equation 6 shows. Note that $\Phi(Y) = 0$ for $\lim Y = -\infty$ and $\Phi(Y) = 1$ for $\lim Y = +\infty$.

$$\Phi(Y) = \frac{1}{1 + e{-Y}} \tag{6}$$

3.1.3 Softmax Activation Function

To continue with, the Softmax Function remakes a vector $\tilde{Y} = (\tilde{y}_1, ..., \tilde{y}_n)$, where $\|\tilde{\mathbf{Y}}\| = n < \infty$, $n \in \mathbb{R}$ where each $\tilde{y}_i \ \forall i \in [1, ..., n]$ belongs to a probability distribution, $\tilde{y}_i \in (0, 1)$ (Saxena 2021*b*). It is worth noting that the Softmax activation function is not only known as a normalized function but is commonly applied at the output layer of a Deep Neural Network.

$$\text{Softmax}(\tilde{y}_i) = \frac{\exp(\tilde{y}_i)}{\sum_{j \in [1,..,n]} \exp(\tilde{y}_j)} \tag{7}$$

3.1.4 ReLu Activation Function

The most widly used active function for Neural Networks and CNNs is Rectified Linear Units (ReLu) Activation function, due to the not heavy prossesing requirments. ReLu returns 0 if the input value is a negative number, and if the number is positive, it returns the given input as the equation 8 indicates (Brownlee 2019*e*).

$$ReLu(Y) = \begin{cases} Y, & Y > 0 \\ 0, & Y \leq 0 \end{cases} \tag{8}$$

3.1.5 Loss Functions & Optimizers

Once the network architecture is determined, the training of a Deep Neural Network is based on optimizing and selecting a crucial measure known as the Loss (or Cost) function and an Optimizer (Brownlee 2019*f*). The loss function is the measure that will be minimized during the learning

process and is defined as the difference between predicted/computed output p of the DNN and the True y value over a vector input x (Abdel-Jaber et al. 2022). Some of the most known loss functions are the Mean Squared Error (MSE) and Categorical Cross-Entropy Loss. It is easier to understand these terms using mathematical concepts as follows.

Let us assume a vector of input, $\mathbf{Input} = (\vec{Input}_1, \ldots, \vec{Input}_n)$, a vector of output of the predicted values, $Activation = (Act_1, \ldots, Act_n)$, and the ground-truth values $y = (y_1, \ldots, y_n)$.

The Mean squared error is then defined as (Calle 2020*a*):

$$MSE(\mathbf{Input}) = \frac{1}{\|\mathbf{Input}\|} * \sum_{i=1}^{\|\mathbf{Input}\|} (Act_i - y_i)^2 \tag{9}$$

and the Cross-Entropy is defined as (Brownlee 2019*b*):

$$L(Activation) = -\sum_{j=1}^{\|\mathbf{Activation}\|} Act_j * \log(Act_j) \ . \tag{10}$$

Loss functions are used to handle different kinds of tasks and produce different types of errors for the same prediction. Thus, a loss function significantly impacts the model's performance. Backpropagation is a method for minimizing the loss function, which is used to create accurate predictions (Agrawal 2017).

Backpropagation is an algorithm that calculates the derivatives of a Neural Network and allows the Deep Neural Network to self-adapt its weights based on the difference between the actual and desired outputs (HANSEN 2019). Thus, optimizers train the network by employing the gradients from backpropagation. Optimizers are either functions or algorithms that adjust the characteristics of the neural network, such as weights and learning rate (Gupta 2021) (a measure of how efficiently the model responds to the situation (Brownlee 2019*g*)). Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) are the most commonly used optimizers.

In more detail, SGD is an optimization algorithm that matches the true gradient values of the loss (cost) function (Calle 2020*b*). It is worth noting that SGD is an optimization algorithm of Gradient Decent (DG), with the key difference between them being that SGD uses only one or a subset of training samples from the training set. Gradient descent (GD) is an iterative process that starts from a random point and descends until it finds the lowest point in a function, searching for a local minimum or maximum of a given function (Bottou et al. 1991). Note that the GD algorithm works

only for differentiable [1] and convex [2] functions (Kwiatkowski 2021). However, to minimize the loss function of a predictive model on a training set, gradient descent requires a significant number of calculations. As a result, stochastic gradient descent is an extension of gradient descent as it drastically reduces calculations by randomly selecting one data point from the whole data set at each iteration (Brownlee 2021*a*).The SGD is determined by considering all training instances, and the estimated gradient is calculated iteratively by taking one training data point each time until all of the training samples have been examined (Brownlee 2017).

On the other hand, Adam is an extension of SGD introduced in 2015 by Kingma. D and Ba. J combine two gradient descent methods, known as Momentum and Root Mean Square Propagation (RMSP) (Kingma & Ba 2014). Adam is a stochastic objective function algorithm that uses first-order gradients and adaptive estimations of lower-order moments (Sanghvirajit 2021). Momentum uses the exponentially weighted average of the gradients is taken into account to speed up the gradient descent procedure. An adaptive learning algorithm called Root Mean Square Propagation (RMSProp) keeps per-parameter learning rates that are modified depending on the average of recent weight gradient orders of magnitude (Ruder 2016).

3.2 Conventional Neural Networks (CNNs)

CNNs (Convolutional Neural Networks) are neural networks that efficiently process input data with a grid-like architecture, such as images, and are developed to learn spatial feature structures from essential to complex patterns (Yamashita et al. 2018). It is relevant to note that CNNs are efficient in tasks such as semantic parsing, image classification, object detection, and natural language processing (NLP) (Thomas 2019).

Thus, this paper uses CNNs for object detection in 2D-Floorplan images, where images are a two-dimensional grid of pixels/values. Convolution layers, Pooling layers, and fully connected layers are the three main types of layers that compose a CNN. Convolution and Pooling layers' main purpose is feature extraction, whereas Fully connected layers (mainly a FFNN) transfer the retrieved features into the output (Mishra 2020).

3.2.1 Convolution and Pooling Layers

To start with, the foundational component of CNN is the convolution layer; it handles the majority of the network's computational load. Convolutional layers are mainly employed for feature extraction, in which a matrix or array of numbers, known as the kernel (or filter), is applied to the image (or input) (Shahid 2019). Each kernel is moved throughout the image's height and width,

[1] Suppose that $f : (a, b) \to \mathbb{R}$ and $c \in (a, b)$. Then f is differentiable if $\exists \lim_{h\to 0} \frac{f(c+h)-f(c)}{h}, \forall c \in (a, b)$ (jenn 2021)

[2] A univariate function f is convex when a line segment connecting two points of the function (x_1 and x_2) lays on or above its curve. $f(\lambda * x_1 + (1 - \lambda) * x_2) \leq \lambda * f(x_1) + (1 - \lambda) * f(x_2)$, where λ denotes a point's location (Minty 1964)

and calculates the dot product between the kernel and the corresponding pixels/values of the image (Mostafa & Wu 2021). This process is continued while applying various kernels to create any number of feature maps that reflect various aspects of the input. The size of a kernel, indicating the filter mask's ($width \times height$) is usually smaller than the input and is typical 3×3.Note that the kernel size is one of the essential hyperparameters for a CNN since it has the potential to reduce or raise the computational costs and weight sharing, which eventually results in lighter weights for back-propagation (MK 2020).

However, padding and stride play also an important role at a convolutional layer since, during the convolutional procedure, an image shrinks and information can be lost (Alto 2020). Thus, when using a convolutional kernel, padding aims to keep an image's original dimensions, and stride refers to the method of squeezing a filter across an input image while increasing the step size (Dumoulin & Visin 2016). Figure 3 summarizes the procedure of a Convolutional Layer with no padding and stride equals to 1.

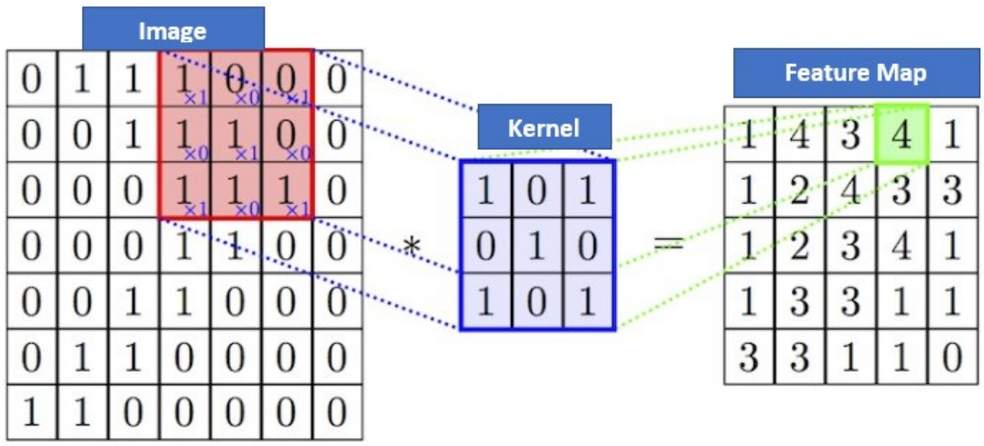

Figure 3: A convonutional Layer's proceudre with kernel zise 3×3, no padding and stride equals to 1.

After the convolutional layer has produced the features map, the downsampling procedure uses a pooling layer. This layer's main goal is to scale down the convolved feature map to reduce the computations, by reducing the connections between layers and working separately on each feature map (Brownlee 2019*d*). The two frequent functions used in the pooling process are Average Pooling and Max Pooling, which calculates the average and the maximum value for each patch on the feature map. Thus, the pooling procedure is then a clarification rather than learning (Saha 2018).

Figure 4: Average Pooling on the left and Maximum Pooling on the right 2×2 layers by sliding a 2×2 window across the input with a step size 2.

A 3-dimensional matrix that is produced from the last pooling or convolutional layer is flattened (meaning that all of its values are unzipped into a vector) and used as the input for a fully connected layer (Nakahara et al. 2017). As Figure 5 illustrates, the architecture of a Convolutional Neural Network from the beginning to the end propose that a completely connected layer is essentially a feed-forward neural network (see Section 3.1). Hence, CNNs are the most applicable models for Object detection, Semantic segmentation, and image captioning.

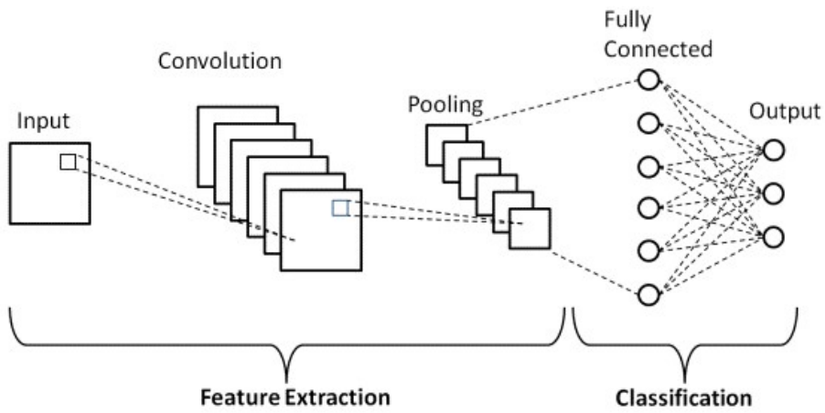

Figure 5: The architecture of a simple Convolutional Neural Network.

3.3 Deep Object Detection

Object detection (OD) is a powerful computer vision technique that focuses on finding and categorizing objects or items inside images or videos, using machine learning or deep learning to comprehend images fully (Zhao et al. 2019). OD is associated with various applications, such as image classification, face recognition, and autonomous driving, and is capable of providing valuable data for semantic comprehension of videos and images (Raghunandan et al. 2018). Object Detection's main challenges are identifying the locations of items in a given image (object localization) and identifying the categories (object classification) to which each object belongs as shown in example in Figure 6 (Brownlee 2019*c*).

Figure 6: Example to identify the difference between Image Classification (left), Object Localization (middle) and Object Detection (right).

In contrast to object localization, image classification forecasts a class label (outputs one or more labels) for an image (given as input) . At the same time, object localization identifies objects/items in a given image (input) and uses one or more bounding boxes to verify their location (out-

put) (Szegedy et al. 2013). The primary idea behind object detection is to determine the existence of items with a bounding box and to identify the categories or classes of those objects in an image. Thus, informative region selection, feature extraction, and classification may be used to split the pipeline of object recognition models into three components (Dickson 2021). Identifying items in digital pictures is one of several related computer vision tasks collectively referred to as "object recognition". Informative Region Selection (IRS) is the process of identifying various item sizes or aspect ratios at any location in the image (Zhao et al. 2019). At the same time, feature extraction and classification relate to the process of separating a target item from all other categories (classification) using visual characteristics that can offer a meaningful and robust representation (feature extraction) (Kibria & Hasan 2017). Lastly, Object detection methods are classified as either neural network-based or non-neural methods. For non-neural methods, it is required to specify features and then classify them using a classification technique (e.g., Support Vector Machine). While, Neural methods can recognize objects without specific features and are often based on deep learning, such as convolutional neural network (O'Mahony et al. 2019).

Network-based models are split into two categories, the one-stage and the two-stage object detection models. One-Stage Object Detection Models are consisting of a category of object detection algorithms that execute detection over a dense sampling of locations instead of skipping the region proposal stage as with two-stage models (Adarsh et al. 2020). Such models are YOLO (Chen & Juang 2022) (the detection algorithm this project use), SSD (Lu et al. 2021), and DetectNet (Tiwari et al. 2022), and it is worth noting that one-stage types of models usually have faster inference . In contrast to one-stage algorithms, two-stage object detection algorithm first develops a region proposal before categorizing and locating the item (Du et al. 2020). At the same time, the most common two-stage methods are Faster R-CNN, R-FCN, and FPN. Overall, one-stage algorithms are faster, and two-stage algorithms are more accurate than one-stage algorithms.

3.4 You Only Look Once (YOLO)

This Section introduces the "You Only Look Once" (YOLO) model, one of the most widespread model architectures and object detection methods. YOLO family falls under the category of one-stage object detection algorithms as we discussed in Section 3.3 and is based on a neural network structure producing high accuracy and speed (Sharma 2022*d*).

3.4.1 History of YOLO family

Joseph Redmon and colleagues introduced the YOLO model in their article "You Only Look Once: Unified, Real-Time Object Detection" in 2015 (Redmon et al. 2016). The method uses a single neural network trained from beginning to end, receiving an image as input and predicting bounding boxes and class labels for each object in the given image. YOLOv1 has an inference speed of 45 frames per second and a mean Average Precision (mAP) of 63.4 (22ms per image). It was far faster than the RCNN family at the time, whose inference rates ranged from 143 ms

to 20 s (Tsang 2018*a*). Joseph Redmon and Ali Farhadi published YOLOv2 in their article titled "YOLO9000:Better, Faster, Stronger" in 2016. The 9000 indicated that YOLOv2 could recognize more than 9000 different types of objects (categories) and had numerous improvements over YOLOv1 (Redmon & Farhadi 2017).

The third version of YOLO was introduced in 2018 by Joseph Redmon and Ali Farhadi in their paper titled "YOLOv3: An Incremental Improvement." Although barely bigger than the prior models, YOLOv3 was still sufficient in terms of speed and accuracy (Redmon & Farhadi 2018). Alexey Bochkovskiy et al. published YOLOV4 in their 2020 work titled "YOLOv4: Optimal Speed and Accuracy of Object Detection". The CSPDarknet53 is the backbones of the YOLOv4 detection model, which dominates other detection models (different of YOLOv3) (Bochkovskiy et al. 2020). Finally, just a few days after the publication of YOLOv4, Ultranytics published YOLOv5, the fifth member of the YOLO family. It is notable that no paper has been published, and there is discussion among the community as to whether or not the PyTorch implementation of YOLOv3 merits the use of the YOLO brand (Jocher 2020). Thus, the technical presentation is paused in the 4^{th} version of YOLO for this project.

3.4.2 YOLOv1

As the first single-stage object detector method to address detection as a regression problem, YOLO represented a milestone in the area of object detection. YOLOv1 employs a single neural network to predict class probabilities and bounding box coordinates from an entire image in one pass due to its detection system, in contrast to other two-stage detectors (Fast RCNN, Faster RCNN) (Redmon et al. 2016).

YOLOv1 Detection Process

YOLOv1 *Detection System* (DS) first resizes the given image to 448×448, then YOLOv1 uses a single convolutional neural network at the image and finally filters the detected objects based on the model's performance (Tsang 2018*b*), as the left panel of Figure 7 summarizes.

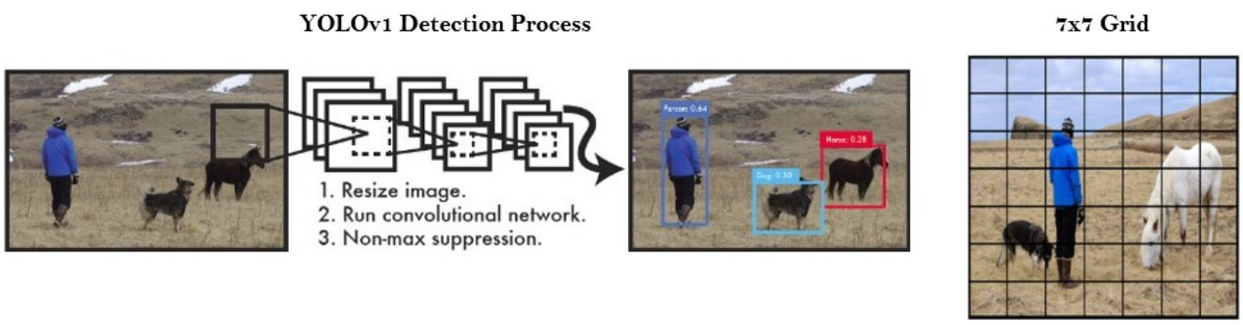

Figure 7: The left part of the Figure summarizes the Detection Process of YOLOv1, according to the original paper (Redmon et al. 2016) and the right part illustrates a 7×7 grid indicating that the image is divided into 49 grid cells.

The input image (or video) is divided into an $S \times S$ grid (see the right panel of Figure 7), where each grid cell controls the detection of an object/item (Karimi 2021); thus, if the object's/item's center falls in the specific grid, this grid is responsible for the detection of the object (Deng et al. 2020). Note that each grid cell may detect B bounding boxes and B confidence scores one for each of those bounding boxes. The model attempts to eliminate $B-1$ bounding boxes in each cell with the lower Intersection over Union (IoU) with the ground-truth box during training. Each box is provided with a confidence score indicating the certainity of the model according to an item in a specific bounding box (Jiang et al. 2022).

According to Joseph Redmon et al., the confidence score can be formulated as follows:

$$C = P_r(\text{object}) \times IoU, \quad \text{where} \quad P_r(\text{object}) = \begin{cases} 1, & \exists\, \text{object}, \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

YOLOv1 uses Non-Max Suppression (NMS) to identify the optimum bounding box among the several predicted boxes and rejects all alternative bounding boxes (Rothe et al. 2014). The objectiveness score provided by the model and the Intersection over Union of the bounding boxes are the initial two factors that the NMS considers (K 2019). The IoU uses a ground-truth bounding box and a detected bounding box to evaluate the detections' accuracy. Figure 8 (left panel) illustrates that IoU is defined as the ratio of the overlap over the union regions (Rosebrock 2016); IoU may be any value in the range of 0 and 1. Thus IoU is equal to zero if two boxes do not intersect, whilst on the other hand, the intersection and union regions are equal if they entirely overlap (Subramanyam 2021). In this situation, the IoU is equal to 1, as the right panel of Figure 8, illustrates. Therefore, the closer to 1 the IoU value, the better the prediction for the specific bounding box.

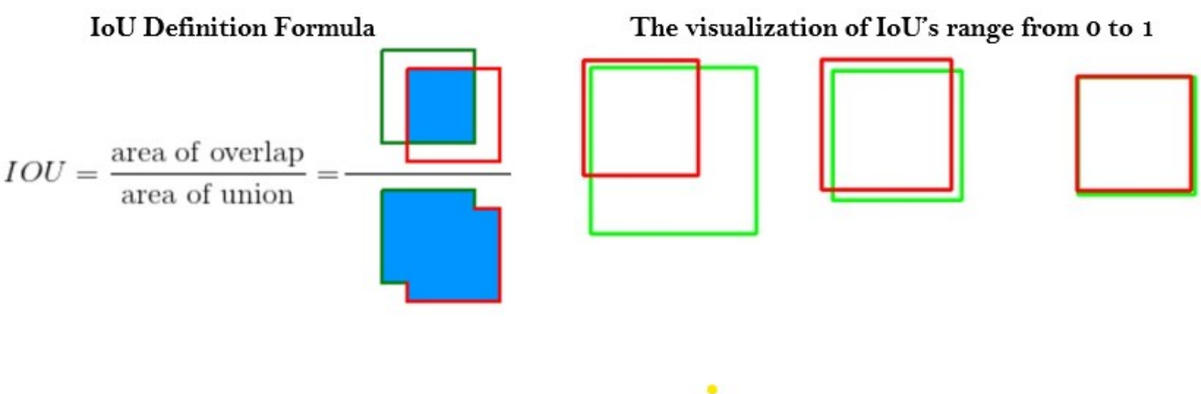

Figure 8: The mathematical formula of Intersection over Union at the left panel of the image and the corresponding range of IoU.

It's worth noting that a bounding box may only contain/predict one item at a time, illustrating a significant drawback of YOLOv1 (Aly et al. 2021). A bounding box contains five parameters as shown in 12.

$$y = (x_{\text{center}},\, y_{\text{center}},\, \text{height}_{\text{box}},\, \text{width}_{\text{box}},\, CS) \tag{12}$$

The vector y includes the x_{center} and y_{center} coordinates of the object's center relative to the origin of the grid cell, the box's $\text{height}_{\text{box}}$ and $\text{width}_{\text{box}}$ relative to the whole image, and the confidence score (CS) given by the network (Kapil 2018*b*). Indicating that $(0, 0) \leq (x_{\text{center}}, y_{\text{center}}) \leq (1, 1)$ and $(\text{height}_{\text{box}}, \text{width}_{\text{box}})$ can be higher than 1; we have $0 \leq \text{height}_{\text{box}} \leq \text{height}_{\text{image}}$ and $0 \leq \text{width}_{\text{box}} \leq \text{width}_{\text{image}}$.

YOLOv1 Architecture

YOLOv1's network design, which comprises Convolutional, Maxpool, and Fully Connected layers, was influenced by the GoogLeNet model used for picture classification. Figure 9 depicts the network's structure, consisting of 24 convolutional layers that extract picture features, followed by two fully connected layers for the prediction of the bounding box coordinates and the classification scores (Kapil 2018*a*).

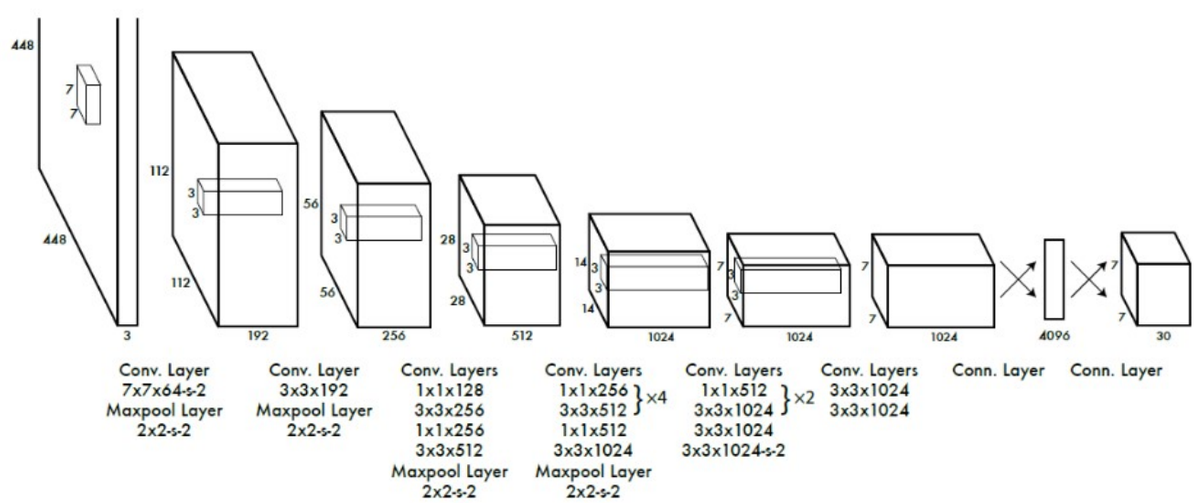

Figure 9: The network architecture of YOLOv1 (Redmon et al. 2016).

The ImageNet 1000-class dataset is used to pre-train the first 20 convolutional layers, which are followed by an average pooling layer and a fully connected layer. Note that the layers consist of 3×3 convolutional layers and 1×1 reduction layers (Ankushsharma 2020). Moreover, the network is trained for object detection by adding the last four convolutional layers followed by two fully connected layers. While the previous convolutional layers utilize ReLU activation, the final layer uses linear activation to predict class probabilities and bounding boxes (Mohan 2020*a*).

3.4.3 Yolov2

The authors claim that YOLOv1 has a relatively poor recall and substantially more localization mistakes than Fast-RCNN (Benjdira et al. 2019). Therefore, they suggest two new YOLO versions, named YOLOv2 and YOLO9000 (Redmon & Farhadi 2017). Both detection algorithms are similar

in terms of design but differ in their training strategies. Thus, according to the authors, YOLOv2 is trained on detection datasets such as Pascal VOC and MS COCO, whereas YOLO9000 is developed to predict over 9000 distinct object categories and is trained on the MS COCO and ImageNet [3] datasets (Ching 2021). Batch Normalization (BN), High-Resolution Classifier, Convolutional With Anchor Boxes, Dimension Clusters and Direct location prediction are the main significant improvements over YOLOv1 (Du 2018) that we discuss one by one in the subsequent text.

Batch Normalization

The normalization process involves converting the numerical value variable to a common scale without distorting contrasts in the value range (Singla 2020). While training deep neural networks, batch normalization is used to normalize the inputs to each layer for each mini-batch. The learning process is stabilized due to batch normalization, and the amount of training epochs needed to train deep networks is drastically reduced (Saxena 2021*a*). According to the authors, the mAP of YOLO increased by 2% by including batch normalization after all convolution layers. Thus, to enhance generality, the network does not need to apply any dropout since batch normalization helps to regularise the model (Brownlee 2019*a*).

High-Resolution Classifier

The network of YOLOv1 was forced to learn object detection while adjusting to the increased input resolution due to the pretraining of the classification step (Sharma 2022*b*). Thus, the network weights had to adjust to the new resolution while still learning the detection, causing a challenge for the network weights. Contrarily, YOLOv2 uses 224×224 (image size) for the pre-training classification phase. The classification network should be fine-tuned using the same ImageNet data for ten iterations at the upscaled 448×448 resolution (Hui 2018*a*). Since the network have already seen that image during the fine-tuning classification stage, it is able to modify its filters to function better on the upgraded resolution. Due to the modification stated above, the mAP was increased by 4% (Benjdira et al. 2019).

Convolutional With Anchor Boxes

The authors of YOLOv2 remove all of the network's fully connected layers and replace them with anchor boxes to predict the bounding boxes (Redmon & Farhadi 2017). To quickly improve the network, anchor boxes, a collection of preconfigured boxes with a certain height and width, act as a prior or estimate for the items in the dataset (Christiansen 2018). Overall anchor boxes are chosen to correspond with ground truth bounding boxes. In addition, one pooling layer is eliminated to improve output resolution, and 416×416 images are utilized for training the detection network. It is essential to emphasize that the model performs 69.5% mAP and recall of 81% without anchor boxes and 69.2% mAP and recall of 88% with anchor boxes, according to the authors (Hui 2018*a*).

[3] Pascal VOC, ImageNet, and MS COCO datasets are large collections of annotated images specifically for computer vision research tasks.

Dimension Clusters

To locate anchor boxes for the datasets, YOLOv2 employed a clever approach. Instead of needing to manually select anchor boxes, the authors choose superior priors that more accurately match the data (Tsang 2018*c*). The authors utilized the IoU between the bounding box and the centroid rather than the Euclidean distance when applying k-means clustering on the training set bounding boxes to locate appropriate anchor boxes.

$$d(\text{box}, \text{centroid}) = 1 - \text{IoU}(\text{box}, \text{centroid}) \tag{13}$$

Therefore, the distance metric was modified by the authors (Redmon & Farhadi 2017) as shown in (13). It is worth notable that using traditional Euclidean distance based on k-means clustering is insufficient since larger boxes produce more inaccuracy than smaller ones (Mohan 2020*b*).

Direct location Prediction

YOLOv1 has no restrictions on estimating the location of the bounding box; since can predict the bounding box anywhere in the image when the weights are initialized arbitrarily (Weng 2018). Thus, the model is unstable during the training process and takes time to forecast meaningful offsets. Using the same methodology as in YOLOv1, but with the addition of the anchor concept, YOLOv2 forecasts position coordinates in relation to the location of the grid cell (Han et al. 2021). However, the algorithm outputs the offsets. Thus, instead of predicting the coordinates as Equation (12) demonstrates, YOLOv2 predicts five parameters t_x, t_y, t_w, t_h, and t_o (indicating the predicted offsets), where the first four are the coordinates of a bounding box and t_o stands for the confidence score according to J.Redmon and A.Farhadi (Redmon & Farhadi 2017). The proposed strategy works since the selected anchor box (using k-means clustering see paragraph 3.4.3) should resemble our ground-truth box. However, the strategy limits the ground-truth between the range of values 0 and 1; thus, the sigma function is applied to limit the range of potential offsets ($\sigma \in [0, 1]$) (Zhang 2020).

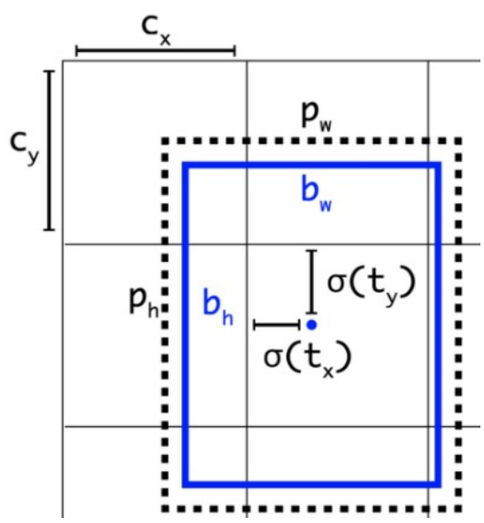

Figure 10: The Bounding box prediction with Direct location, where the center of the bounding box lies in the second row, this cell is in charge of foretelling the existence of the blue box.

Figure 10 displays the anchor box and the expected bounding box (with black lines), whilst at the same time the center of the associated estimated bounding box b is indicated by (b_x, b_y), its width by b_w, and its height by b_h. The anchor box size is (p_w, p_h) and is located at (c_x, c_y) in the top-left corner of each grid cell (Kezhuang 2020). The predictions match the following formula if the cell is offset by (c_x, c_y) from the image's top left corner and the bounding box has the dimensions (p_w, p_h) (Li et al. n.d.).

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x, \\
b_y &= \sigma(t_y) + c_y, \\
b_w &= p_w \times e^{t_w}, \\
b_h &= p_h \times e^{t_h}, \\
\sigma(t_o) &= \Pr(\text{object}) \times \text{IoU}(\text{bounding box}, \text{object})
\end{aligned}
\tag{14}
$$

Because of this direct location limitation, the mAP of YOLOv2 was increased by 5%.

Darknet-19

YOLOv2 also introduces a new backbone named Darknet-19 as a point of improvement. The network is made up of 5 max-pooling layers and 19 layers of convolution, as shown in Figure 11. Note that the authors compressed the feature representation between convolutions using filters rather than fully connected layers for prediction (Setiawan & Purnama 2020).

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 11: Darknet-19 Architecture.

3.4.4 Yolov3

YOLOv3 was published in 2018 in the paper "YOLOv3: An Incremental Improvement" as an improved version of YOLO and YOLOv2 by Joseph Redmon and Ali Farhadi. The architecture of YOLOv3 is identical to that of YOLOv2 (Redmon & Farhadi 2018); however, the authors include several design modifications, such as adding the Darknet-53, a new Backbone, switching out the softmax function for discrete logistic classifiers, and detection of small items with across scales.

Darknet-53

Darknet-53, the feature extractor YOLOv3 employs, was initially a 53 layers network (see Figure 12) that had been pre-trained on the Imagenet dataset (Bandyopadhyay 2022). Nevertheless, 53 more layers are added to it for the purpose of detection, giving us 106 layers of fully convolutional layers and residual blocks as the base architecture for YOLOv3. As a result, YOLOv3 could run 30 frames per second (FPS), indicating a slower compared to YOLOv2 (Sharma 2022*c*).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 12: Darknet-53 Architecture.

The residual block is a block of convolutional layers designed to address the degradation problem (Sandhu 2020). The degradation problem stands for the accuracy issue that might develop in a huge deep neural network as more layers are added (Feng 2017). The primary concept behind a residual block is that each layer's output gets added to a layer further down in the block. It is noteworthy that the number of network layers increases significantly without information loss when a residual block is present (Oommen 2020).

Multi labels prediction or Discrete Logistic Classifiers

In earlier versions of YOLO (YOLOv1 and YOLOV2), authors utilized softmax to classify objects

and determined that the object within the bounding box belonged to one class, the one with the highest score (Kathuria 2018). As a result, the first versions of YOLO could only recognize one class per bounding box. In contrast, YOLOv3 treats the class prediction as a multilabel classification issue (i.e., the output label can be “man” and “person”), which implies that the class labels are considered mutually non-exclusive (Kamal 2019). This was accomplished by employing sigmoid or logistic classifiers to predict the class labels for each bounding box (Redmon & Farhadi 2018). During training, YOLO v3 calculates the classification loss for each label using binary cross-entropy (see equation 10, with the modification that the input and predicted values are binary), whereas class predictions and object confidence are predicted using logistic regression (Chakure 2021).

Predictions Across Scales

Previous YOLO versions had trouble detecting small objects since the detection only happened at the last layer (Sharma 2022*c*). However, in YOLOv3, as seen in Figure 13, the objects are recognized at three separate network levels, at layers 82, 94, and 106 (Mantripragada 2020). Feature pyramid networks, a related idea, served as the inspiration for the feature extraction from each of the three scales (FPN). The feature extractor Feature Pyramid Network (FPN) creates many feature map layers in place of detectors like Faster R-CNN (Hui 2018*b*).

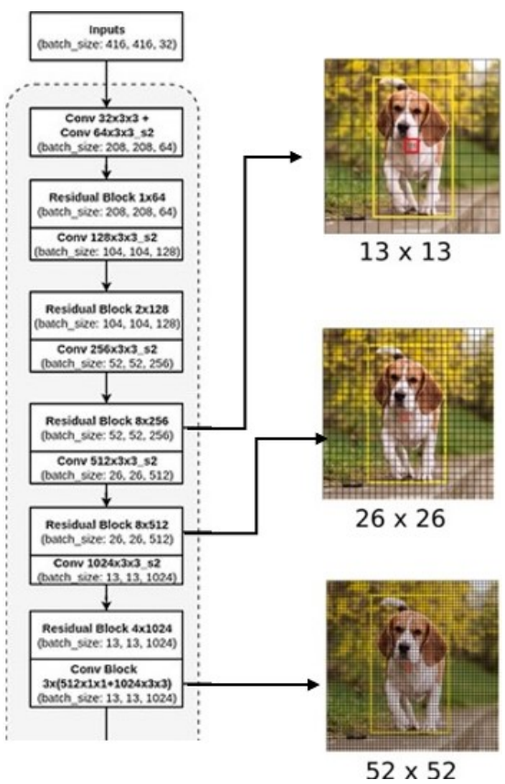

Figure 13: Object detection with the 82, 94, and 106 detection layers.

Feature maps of three distinct sizes, with strides of 32, 16, and 8, are detected using the detection layer (Balsys 2019). For instance, if the input image has a size of 416×416 pixels, the detections are done at scales of 13×13, 26×26, and 52×52, as calculated by "image size/stride", for strides $32, 16$ and 8 respectively (see Figure 13). Finally, the detection is carried out by applying 1×1 detection kernels to feature maps the three different sizes at three various locations throughout the

network (Zhao & Li 2020).

3.4.5 Yolov4

Alexey Bochkovskiy et al. introduced YOLOv4 in their paper published in 2020 with the title "YOLOv4: Optimal Speed and Accuracy of Object Detection" ; which increases YOLOv3's mAP (mean Average Precision) and FPS (Frames per second) on the MS COCO dataset by 10% and 12%, respectively (Bochkovskiy et al. 2020). During training, YOLOv4 tweaks contemporary architectures and employs the " Bag of Freebies " and " Bag of Specials " approaches that assist in improving the training strategy and the detector's accuracy, respectively (Cohen 2020). It is important to note that the YOLOv4 architecture employs a backbone; responsible for predicting bounding boxes and class labels, a neck; composed of various top-down and bottom-up pathways, and a head; feature extractor (Sharma 2022*a*).

Backbone

CSPDarknet53, a convolutional neural network that uses Darknet-53 for object identification, is defined as the backbone used by YOLOv4 (Tsang 2021). Cross Stage Partial is denoted by the letter "CSP." The fundamental goal of CSP is to aggregate the results by splitting the current layer into two components: one that will go through a block of convolutions and one that will not (Wang et al. 2020). The primary goal of adopting CSPDarnet-53 as the backbone is to allow this architecture to obtain a richer gradient combination while requiring less computing (Solawetz 2020).

Neck

The neck is defined as the layers that are often inserted between the back and the head to gather feature maps at various stages (Choudhary 2021). A neck is primarily designed to improve the information that enters the head (Trivedi 2020*a*); as a result, a neck is made up of both top-down and bottom-up paths. According to the authors YOLOv4 employs the Path Aggregation Network (PANet) and the Spatial Pyramid Pooling (SPP) as the neck. According to Liu et al., Path Aggregation Network intends to improve information flow in a proposal-based instance segmentation architecture (Liu et al. 2018). To correctly maintain spatial information, PANet is chosen, for example, in YOLOv4 segmentation (R 2020). SPP-Net, a convolutional neural network that uses spatial pyramid pooling, is used to widen the receptive field and distinguish the most crucial characteristics from the supporting features (He et al. 2015).

Head/Dense Prediction

Bounding boxes are located, and each box's category is classified using the head block. As Section 3.3 highlights, the head is often divided into two categories; the one-stage object detector and the two-stage object detector. By employing the same procedure, YOLOv4 utilizes YOLOv3 as the head. The network determines the confidence score for a class as well as the bounding box coordi-

nates (x, y, w, h). Figure 14 displays a summary of YOLOv4's final architecture.

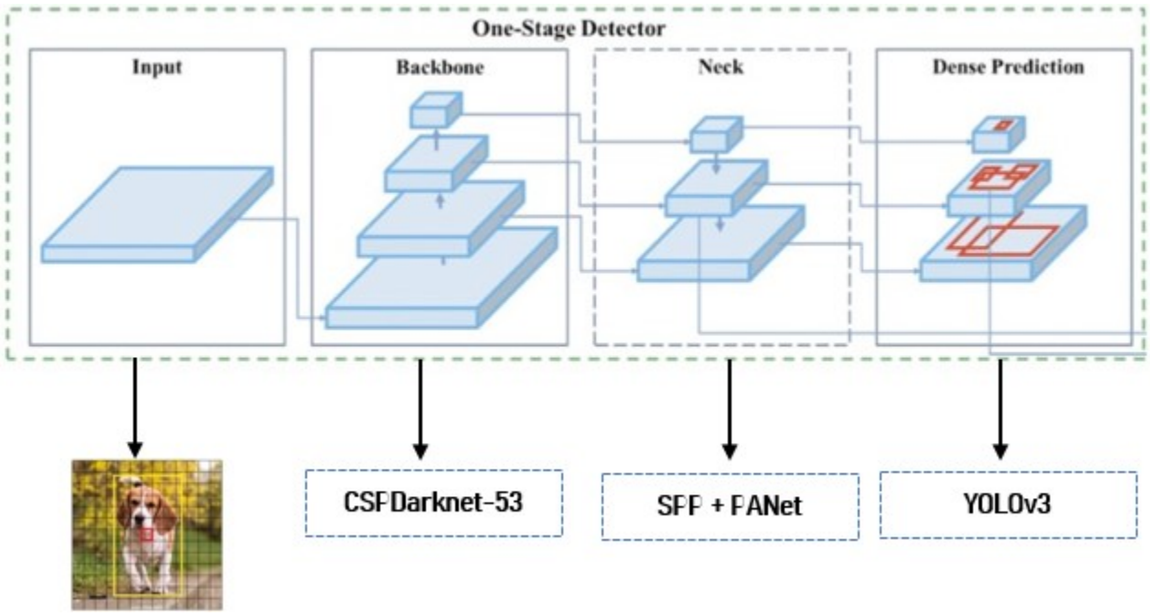

Figure 14: YOLOv4's backbone, neck and head.

BoG and BoS

The authors included two novel optimization techniques named Bag of Freebies (BoG) and Bag of Specials (BoS) in addition to other selected approaches in architecture designs to increase the model's accuracy and be utilized for training and inference (Trivedi 2020*b*).

The term "Bag of Freebies" refers to a group of techniques that alter the training strategy or training costs for improving model accuracy while having no effect on inference speed (Shah 2020*a*). The authors of the Yolov4 list the Data Augmentation, Semantic Distribution Bias in Datasets, and the Objective Function of Bounding Box Regression as the training techniques for object identification (Bochkovskiy et al. 2020).

"Bag of Specials" are methods that can significantly boost model accuracy while just slightly increasing inference cost (Trivedi 2020*c*). Increasing the receptive field is one of these strategies; for example, SPP is used as an additional module in YOLOv4 to widen the receptive field. Moreover, Attention Modules (convolutional neural networks that focus the network precisely on the objects instead of the whole image) and feature integration techniques like skip-connections are some methods of Bag of Specials (Chitale et al. 2020).

3.5 Active Learning

The need for approaching methods that can optimize the time, speed of training, and prediction and improve the model's accuracy is a crucial concept for Machine Learning (ML) and Deep Learning (DL). Obtaining labeled data for new tasks is difficult and expensive. Active learning is one of

the methods being researched and used to enhance the performance of models. It can boost the labelling of new data by choosing unlabeled samples estimated to significantly improve the model's speed and accuracy after labelling.

In Machine Learning (ML), supervised and unsupervised learning are the two primary learning categories. In supervised learning, the algorithm uses data points humans have labeled to create predictions (Ravi n.d.). In contrast, unsupervised learning employs algorithms to evaluate and cluster unlabeled datasets; as a result, the algorithms identify hidden patterns or data groupings without the assistance of humans (Peixeiro 2019). An area of machine learning, active learning, is precisely a type of semi-supervised machine learning. Semi-supervised learning combines both a large number of unlabeled instances and a limited number of samples that have labels (Brownlee 2021*b*). Figure 15 summarises the main categories of Machine Learning and illustrates the difference between the different categories of learning methods.

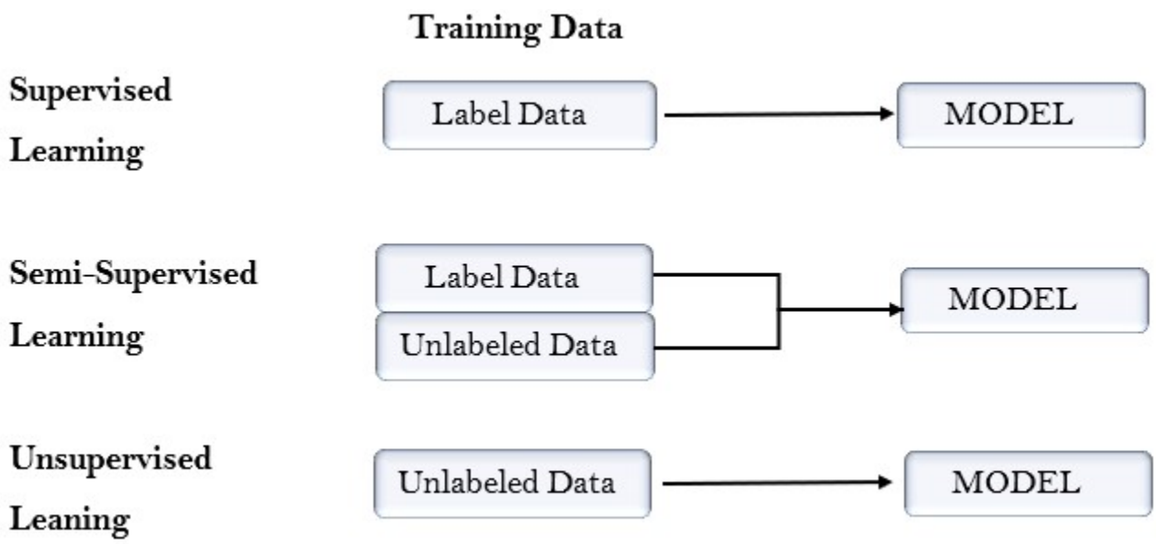

Figure 15: A summary of the main Machine Learning categories.

A Simple Example of Active Learning

The notion that not all data points are equally essential for training a model serves as the foundation for active learning. Active learning systems provide questions in the form of unlabeled instances that need to be labeled to get around the labeling bottleneck. By proactively finding high-value data points in unlabeled datasets, the algorithm can train quicker with the objective of achieving high accuracy with a minimum number of labeled data. The following example was inspired by the book "Active Learning Literature Survey" published by Burr Settles (Settles 2009).

Let us consider a cluster/dataset named C that consists of two sets called A and B and a decision boundary in between, as the left panel of Figure 16 depicts. Notably, the union of A and B ($A \cup B = C$) is equal to the dataset/cluster. Assuming that cluster C contains more than ten thousand unlabeled data points ($\|\mathbf{C}\| < \infty \implies \|\mathbf{A}\|, \|\mathbf{B}\| < \infty$), the model can not learn from this cluster/dataset. Manually labeling each of those points would be time-consuming or very expensive.

Note that Figure 16 only exhibits 500 samples for visualization purposes, with 25 samples in each set. The middle panel of Figure 16 illustrates the results of a model by selecting a random subset for labeling, demonstrating that the trained model performs very poorly in comparison with the right panel, in terms of identifying the optimal bounding line, due to the informativeness of the selected samples.

Figure 16 demonstrates the selection of more informative samples that aid the model's selective learning and provide more promising results. The selection of the most informative samples is the primary goal of active learning, which is how the idea of active learning started and evolved.

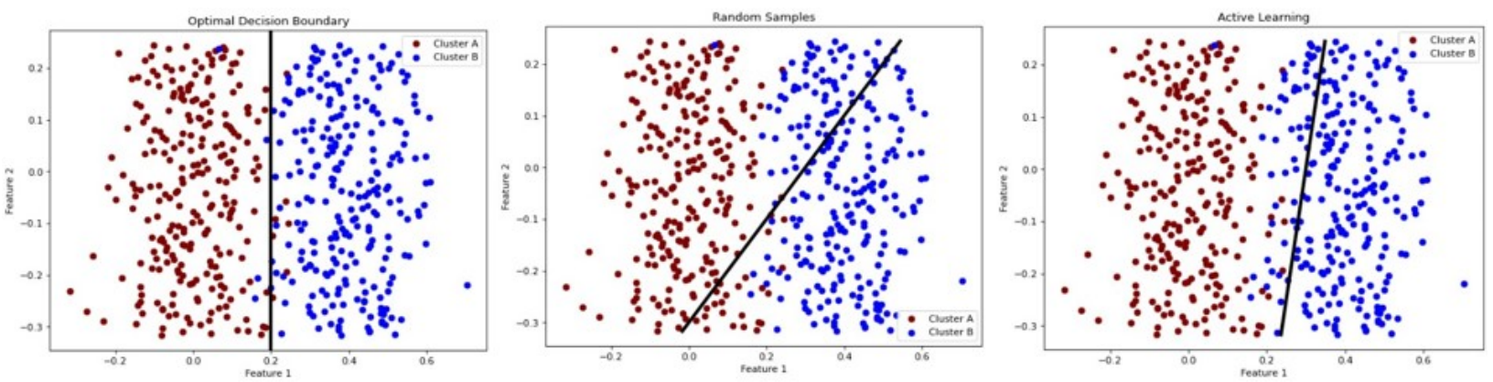

Figure 16: The samples of the Active Learning example are generalized by the Gaussian distribution, such that $X_1 \sim \mathcal{N}(\mu = 0, \sigma = 0.1)$ and $X_2 \sim \mathcal{N}(\mu = 0.4, \sigma = 0.4)$.

Passive Learning Vs Active Learning

The central premise of active learning is the training of a learning algorithm using far less training data and resulting in higher performance than the conventional approaches; by selecting the informative data samples (Cohn 2010). One of the most traditional techniques is passive learning (Cormack & Grossman 2014), which involves passing a sizable amount of labeled data to the learning algorithm. Thus, labeling the entire dataset needs a sizable amount of work, and gathering labeled data is one of the most time-consuming passive learning processes. As a result, large-scale labeled data collection can be hampered in various situations by limiting variables.

3.5.1 Scenarios

Active Learning is just an approach for researching how to label as few examples as feasible while achieving the greatest improved performance. To generate the training set, an Active Learning algorithm attempts to identify the samples with the highest information value by using query methods and measurements (Solaguren-Beascoa 2020). This presupposes that different samples in the dataset have varying values for the current model update. Although the numerous specialized query strategies, such as least confidence, margin sampling, entropy sampling, and many others. This project took into consideration the following three general scenarios (i) membership query synthesis, (ii) stream-based selective sampling, and (iii) pool-based sampling (Cohen 2018). No-

tably, all three cases presuppose that the algorithms will request annotators (humans or machines) to identify/label the unlabeled instances (Prendki 2020).

For generalization purposes, let us assume that D is the dataset we consider. Therefore $D_{K,i}$ are the known data points, $D_{U,i}$ are the unknown data points, and $D_{S,i}$ the selected data points, where $i \in [1, 2, \ldots, \|\mathbf{D}\|]$.

Membership Query Synthesis

Learning using membership queries, proposed initially by Dana Angluin in the paper titled "Queries and Concept Learning" in 1988, is one of the first active learning strategies that are investigated (Angluin 1988). As the name implies, Membership Query Synthesis is a method of creating/synthesizing queries based on a set of membership criteria. In this case, the learner/model can create a new instance from scratch (which might or might not exist) that satisfies the instance space definition because the learning model is aware of the concept of the instance space (Dave 2020). Then a human annotator (oracle) continues by adding the labels. A new artificial query can be created from scratch using a small quantity of labeled data, while Query synthesis can generate examples based on training instances for maximum efficient learning (Nelson 2020). Consequently, the active learning algorithm can decide what would be advantageous (Schumann & Rehbein 2019); for instance, the learning algorithm can build a subset of an image in the labeled training data and use that newly created subset image for additional training. Despite that, a significant issue is that the synthesized queries are frequently meaningless, making it challenging for annotators (humans) to assign labels (Sun & Wang 2010).

Stream-based selective sampling

The central presumption for stream-based selective sampling is the availability of obtaining unlabeled instances ($D_{U,i}$) efficiently, which allows the learner to test the data sample against the actual distribution (D) before deciding whether or not to request its labeling (Nelson 2020). Stream-based selective sampling, also known as "sequential" sampling, involves presenting unlabeled examples sequentially from the data source (one sample at a time, $D_{U,i}$, $\forall i$) to the active learner algorithm for querying, with each query decision being made separately (Dave 2020). The learner must determine if the instance is informative enough for the expert (annotator) to annotate it or if the instance should be avoided. Accordingly, there are various ways to structure the choice of whether or not to query an instance. One method is to analyze samples using a "query strategy" or "informativeness measure" (see Section 3.5.2). Consequently, stream-based sampling must establish a minimum limit for the informative evaluations of the unlabeled instance (Shah 2020*b*); if the instance analysis is more significant than this threshold, the instance will be queried and labeled, and the specific sample will be defined as labeled ($D_{K,i}$). Nevertheless, if the input distribution is uniform, selective sampling may behave like membership query learning since the data's distribution concerns no

assumptions (Sun & Wang 2010).

Algorithm 1 Stream-based Sampling General Algorithm

Require: The AL model is trained on a subset T such that $T \subseteq D$, thus $T_{K,t} \subseteq D_{K,i}\ \forall t \in [1, \ldots, \|\mathbf{T}\|]$

Require: A threshold b based on informativeness measure

1: **for** $\forall i \in [1, \ldots, \|\mathbf{D_{U,i}}\|]$ **do**
2: **if** $i \geq b$ **then**
3: The oracle labels the sample i, thus $D_i \in D_K$
4: **else**
5: Ignore the sample i and move to the next one $(i + 1)$
6: **end if**
7: **end for**

The concept of stream-based selective sampling has been investigated in several practical activities, including Natural Language Processing (NLP) (Tran et al. 2017) and Sound classification/Speech Recognition (SR) (Han et al. 2016).

Pool-based sampling

The key idea of pool-based active learning is to rank the dataset (D_i) samples in some manner; consequently, the algorithm tries to analyze/evaluate the whole dataset before choosing the optimal query or combination of queries (Wu 2018). The next step is to identify the best subset of the unlabeled data ($D_{U,i}\ \forall i \in T$, where $T \subseteq D$) to be labeled so that a model is trained on the most informative samples (Yang et al. 2018) and to label the remaining dataset samples as accurately as possible. Notably, the Active Learning model can concentrate on many data samples simultaneously, in contrast to stream-based selective sampling (Ren et al. 2021). Thus, based on sampling techniques or informativeness metrics, the most informative data samples are chosen from the pool of unlabeled data samples. The following sample to be queried is chosen using this new information after the most informative samples are determined (Gissin & Shalev-Shwartz 2019). The recently queried sample is then added to the labeled set ($D_{U,i} \Rightarrow D_{K,i}\ \forall i \in T$), and training is then carried out. Until the labeled dataset is finished or the pre-defined termination requirements are met, this process is repeated (the main idea of pool-based sampling is present in Algorithm 2 with more details). The primary distinction between stream-based and pool-based active learning is that the former sequentially scans the data and makes query judgments on an individual basis (Sun & Wang 2010), whilst the latter assesses and ranks the whole collection before choosing the most appropriate query, as can be seen in Figure 17. Even though pool-based sampling is more effective than stream-based selective sampling, it necessitates a lot of computing and memory resources. However, pool-based sampling is the most common learning scenario for active learning algorithms in real-world issues (John 2022).

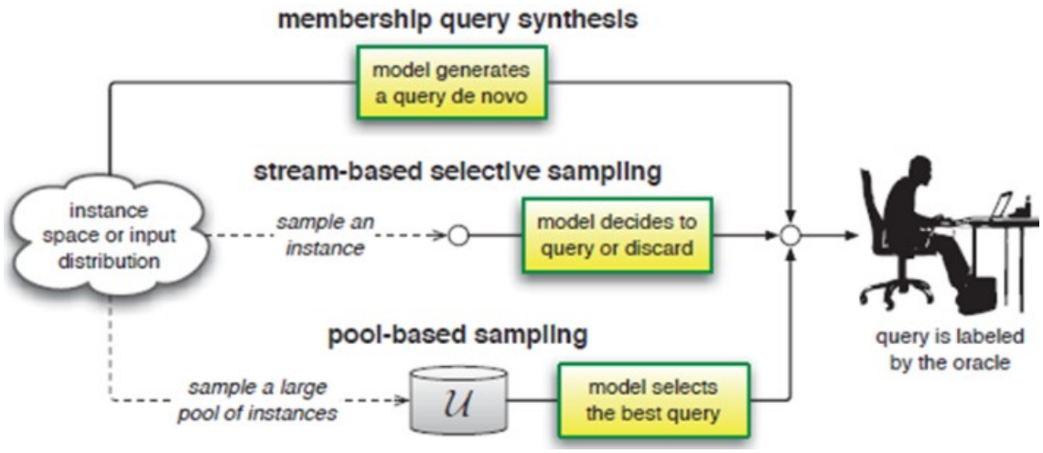

Figure 17: A summary of the three main scenarios of Active Learning.

Figure 17 summarizes the three main scenarios of Active Learning, as discussed in the Section 3.5.1 while Algorithm 2 demonstrates the central concept of a pool-based sampling scenario in Machine Learning in more detail.

Algorithm 2 Pool-based Sampling basic algorithm for a ML model

Require: Split the Dataset D into Pool-data (U) and test data T, where $\|\mathbf{D}\| \gg \|\mathbf{T}\|$

Require: The Pool-data U is divided in to n-samples where $n \in [1, 2, \ldots, \|\mathbf{U}\|]$ for train the model (M), and the remaining $n - \|\mathbf{U}\| (\in \mathbb{R})$ are used for the validation of the model.

Require: Define a threshold t to identify uncertainty (uncertainty measures in Section 3.5.2)

1: Normalize the datasets $(training, validation \subseteq U)$
2: Train model M, with the n-samples
3: Validate $M \rightarrow$ probability measures (p)
4: Test $M \rightarrow$ performance measures (q)
5: **if** $p \geq t$ **OR** $p \leq t$ **then**
6: Move the samples (U_{new}) with higher or lower uncertainty to the training set. $\Longrightarrow$ $training \cup U_{new} = training_{update}$
7: Label the samples.
8: Inverse normalization
9: **if** Stopping criteria **OR** $U = \emptyset$ **then**
10: Exit the procedure
11: **end if**
12: **end if**
13: Retrain M with the $training_{update}$
14: Go to Step *3*

The pool-based scenario has been examined in various machine learning application areas, including text classification (NLP) (Nigam & McCallum 1998) , image classification (Computer vision) (Joshi et al. 2009) and cancer detection (Liu 2004).

3.5.2 Query Strategies

Evaluation of the informational value of unlabeled examples occurs in every active learning scenario. The training examples that can effectively improve the learning performance of machine learning models are found using query strategies. To enhance random sample selection, several studies have proposed and described several AL query strategies, including (1) uncertainty sampling, (2) expected model change, (3) query by committee, (4) expected error reduction, and (5) expected model change. However, the analysis of the Uncertainty Sampling Query Strategies will be the exclusive emphasis of this subsection (Kumar & Gupta 2020).

Note that the predictions are a probability distribution; therefore, each prediction ranges from 0 to 1, and they all sum up to 1. As a result, the prediction vector for n-predicted labels can be defined by $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_n)$, where the two most certain predictions are $\hat{y}_1$ and $\hat{y}_2$.

Uncertainty Sampling Query Strategies

Least Confidence is defined as difference between the most certain prediction ($\hat{y}_1$) and 1:

$$LC = \frac{c * (1 - \hat{y}_1)}{c - 1},$$

where c indicates the number of classes.

Entropy The average amount of "uncertainty" that a variable's potential outcomes contain, can be define mathematically as:

$$Entropy = -\frac{\sum_{t=1}^{n} \hat{y}_i * \log_2 \hat{y}_i}{\log_2 n}$$

Margin Sampling is defined as the variation between the top two most certain predictions:

$$MS = 1 - (\hat{y}_1 - \hat{y}_2)$$

.

Ratio of Confidence is defined as the proportion between the two most certain predictions:

$$RC = \frac{\hat{y}_2}{\hat{y}_1}$$

4 Methodology

As mentioned in Section 3.5, one of the semi-supervised techniques utilized for Object Detection (OD) and image classification using deep learning models is Active Learning (AL). Active learning can increase the efficiency of classifying new data, by choosing the unlabeled samples that, once labeled, are expected to improve the model performance. This Section demonstrates an Active Learning approach and aims to provide a theoretical and practical framework for understanding the relationship between Active Learning and Object Detection for identifying doors, windows, and rooms/zones, in 2D floorplans images.

4.1 Prepossessing Steps

This project uses a small dataset of 500 2D-floorplan images, of which 400 are manually annotated with an image annotation tool named "LabelImg". Data annotation labels data to have the outcome that the deep object detection model requires to make predictions. Hence, data annotation makes labeling, tagging, transcribing, or processing a dataset with features to train a deep learning system to learn and recognize. The labeling annotation tools enable object detection models to enhance computer vision accuracy. Consequently, machine learning models can recognize patterns before they can recognize objects on their own by being trained with labels, usually defined as the "ground truth" labels. Due to the ability to automatically save the images into YOLO format, the LabelImg annotation tool was chosen. Figure 18 demonstrates the procedure of annotating one image and the expected annotation provided as a "txt" file.

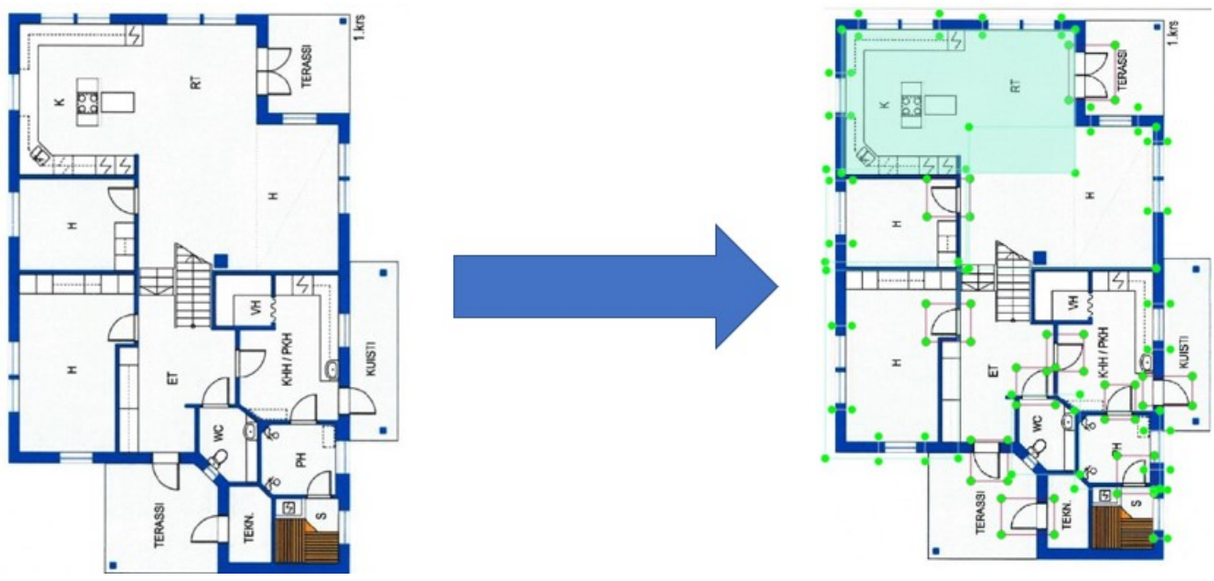

Figure 18: A floorplan image with (right) and without (left) annotations.

As explained in Section 3.4, YOLO object detection family models identify objects based on the bounding boxes; consequently, YOLOv5 anticipates annotations for each image in the form of a text file with each line describing a bounding box, as Figure 19 depicts.

```
0 0.335897 0.174076 0.054212 0.087413
```

Figure 19: The annotations coordinates.

Thus, in Figure 19, the first column indicating the number 0, stands for the class ID, in this case it corresponds to a door; the second column equals 0.335897 is the x-axis value, 0.174076 is the y-axis value, 0.054512 is the width of the object and 0.087413 corresponds to the height of the object. Note that all the given coordinates are with respect to the image size.

Structure of Data Directories

The data is provided in the following structure to adhere to Ultralytics directories.

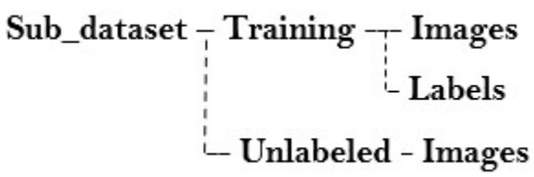

Figure 20: The structure of the dataset.

It is worth mentioning that the "Unlabeled" file can be replaced by a "test " file and an additional sub-file with the test image's annotations in the structure for a traditional object detection method with YOLOv5 based on a custom dataset. However, this project does not imply a test file since it is out of this thesis's purpose. Consequently, this thesis provides only a validation file, indicating the unlabeled images (the creation of the file directories can be found in Section A.1).

The main directory has two files, including the crucial data.yaml, and one sub-directory corresponding to the training and validation datasets, as demonstrated in Figure 20. The data.yaml file contains information about the configurations, including the main path, the paths for the train and validation data directories, the total number of classes in the dataset, and the name of each class, which are illustrated in Figure 21 as path, train, val, nc, and names, repsectively. The YOLOv5 architectures that Ultralytics supports primarily differ in terms of size as follows: YOLOv5n corresponds to nano, YOLOv5s to small, YOLOv5m to medium, YOLOv5l to large, and finally YOLOv5x to extra large. Therefore, due to the small number of images, we conclude training YOLOv5s with some modification at the model's hyperparameters (more details can be found in Section 5.2).

```
path: ./data
train: images
val: images
test:  # test images (optional)

# Classes
nc: 3  # number of classes
names: [ 'Door', 'Window', 'Zone' ]  # class names
```

Figure 21: The data.yaml file.

Since YOLOv5's pre-trained weights are based on the COCO dataset, which comprises real-life objects including persons, bikes, dogs, and more, our model is trained from scratch for 300 epochs with a 16 batch size (the code and more details for training YOLOv5 is provided in Section A.2). Once the model is trained, this thesis uses an Active Learning approach to identify the most informative images.

4.2 Active Learning Methodology

We examine a pool-based sampling approach for active learning in object detection, using the YOLOv5 algorithm trained on 80% of the dataset, indicating 400 annotated images. Our approach aims to identify and cluster the most informative images over a pool of 100 unlabeled images to annotate and retrain our model. In contrast to the standard Active Learning approaches, our approach predicts the bounding box and the label for each object in the image, similar to a standard train and detect prediction process. However, the model's predicted bounding boxes and annotations are defined as the annotations taken from an oracle; thus, our algorithm is automated/autonomous. Consequently, we employ the Least Confidence measure (LC) to identify and cluster the images based on their confidence and retrain our model based on the created clusters. Even if the idea of an automated Active Learning model seems promising, we face a few challenges in employing an automated pool-based sampling technique for active learning, such as

(i) How to calculate the overall score for each image,

(ii) What should be the identical amount of images in each cluster and how many clusters are identical, and

(iii) Based on what measure we have to create the cluster since the implementation of $K-NN$ clustering was impossible due to the limited information.

We address the first issue by defining a value named $b(x)$, the value or measure to evaluate the sample/image x. Each image may contain various distinct items; thus, the b-score for a specific image is calculated as the average of the Least confidence scores for all the bounding boxes in the image. More specifically, let us assume an image with n bounding boxes, where $n \in \mathbf{R}$, and $\|\mathbf{n}\| < \infty$. The primary step is calculating the Least Confidence for each bounding box/ object in the image.

$$LC\text{score} = \frac{num_{class} * (1 - confidence)}{mun_{class} - 1} = \frac{3 * (1 - confidence)}{2}, \tag{15}$$

where num_{class} is the number of classes; in our case is equal to 3. Note that according to the formula of LC in (15), the range of the LC score starts from 0 to 1.5, indicating that if the LC score is closer to 0, then the model is more certain for the specific bounding box; in contrast, if the LC score is closer to 1.5 indicates the uncertainty of the model. To give you an example, if the confidence score for a bounding box is 0.8, then the value of LC score is equal to 0.3 (see equation 15), indicating that the model is certain for the specific bounding box. Once the calculations for all bounding boxes is done a vector is created such that $LC_b = (LC_{b1}, \ldots, LC_{bn})$. Then, the b-score for the image is estimated as the average of the LC vector, as shown in (16).

$$b_{avg} = \frac{\sum_{i \in [1,\ldots,n]} LC_{bi}}{n} \tag{16}$$

Even though the b-score can be defined as the maximum of the LC vector, $b_{max} = \max_{i \in [1,\ldots,n]} LC$, or either the sum of the LC score, $b_{sum} = \sum_{i \in [1,\ldots,n]} LC_{bi}$, our project employs that the b-score is derived by the average.

The second milestone in our approach is to cluster the 2D floorplans image based on their calculated b-scores. Note that the number of images in each cluster must be similarly divided. To resolve these issues, we defined a threshold t as the parameter that controls the number of images in a cluster and simultaneously divides the unlabeled images into three clusters based on their b-score. Our experiments demonstrate that setting $t < 0.6$ create a cluster namely "High Confidence," which is made up of the images with the highest confidence; then setting $0.6 < t < 0.75$ defines a cluster namely "Mid Confidence," which has the images that the model is not very certain and the same time not very uncertain. Finally, the images with $t > 0.75$ made up the "Low Confidence" cluster, which contains the image the model is unsure for the predicted annotations.

This classification system helps distinguish the three main clusters used to retrain the model. It is worth noting that overall, 43 and 41 images have clustered with high and low confidence scores, respectively. However, in the cluster with low confidence is assigned only 26 images due to the limitation of the dataset. Figure 22 provides a general overview of the clusters, having the x and y axis to represent the images IDs and b-scores, respectively. The selection of the specific thresholds is a result of our experimentation; thus, threshold t can be modified based on the model's performance on each dataset. Moreover, due to the low impact of the model in the "low confidence" cluster we prefer not to select many of the automated annotated images.

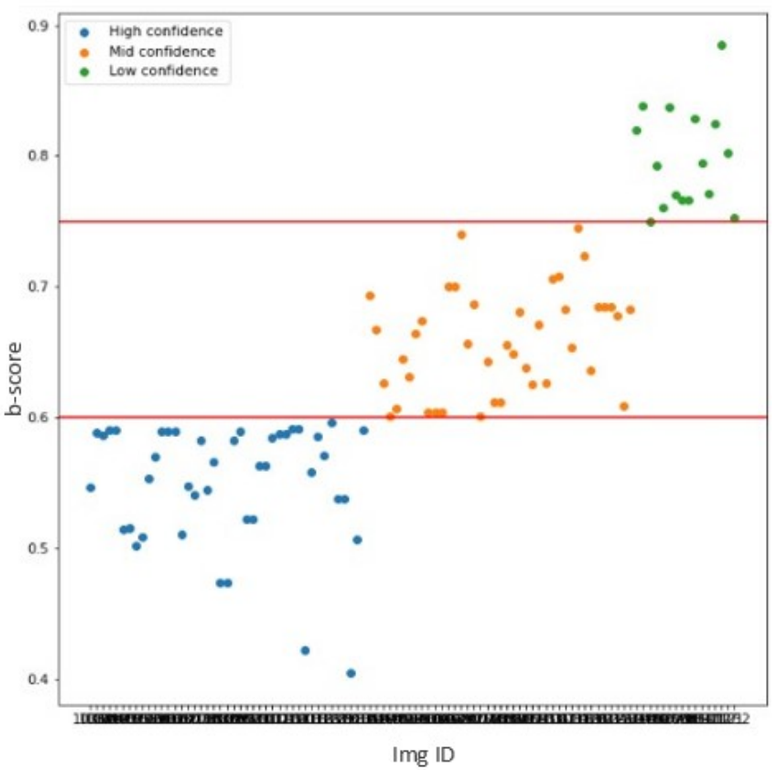

Figure 22: The three clusters based on their b-score.

Algorithm Procedure

The proposed pseudo-code is described in this paragraph, whereas the code is available in Section A.3.

Algorithm 3 Active Learning Algorithm

Require: The YOLOv5 model (M) pre-trained on the 400 floorplan images

Require: The Unlabeled pool of the 100 images (x) is defined as U

for $x \in U$ **do**
  Predict the bounding boxes for each x
  for $bb \in x$ **do**
    Calculate the Least Confidence per bb
    Store the LC score in a list
  end for
  Calculate the Average Least Confidence ($ALC = b_{score}$) per x
  if $b_{score} < 0.6$ **then**
    Store the x in "High Confidence" Cluster
  else if $0.6 < b_{score} < 0.75$ **then**
    Store the x in "Mid Confidence" Cluster
  else
    Store the x in "Low Confidence" Cluster
  end if
  Move the Clusters in the training set

16: **end for**

17: Convert the predicted annotations to YOLO format

18: **for** High Confidence Cluster, Mid Confidence Cluster, Low Confidence Cluster **do**

19: Re-train Model (M)

20: **end for**

5 Experimental Results

The outcomes of the technique outlined in Section 4 are discussed in the subsequent text. In this section, we demonstrate the utility of our approach for object detection in the dataset of 2D floorplans. Our dataset is divided into two portions, which we estimate to be the training set (80%) and the unlabeled pool (20%). The model's annotations and evaluation metrics will then show us how our active learning strategy performed.

5.1 Evaluation Metrics

The precision, recall, and confusion matrix are measurements that will be introduced for the problem of object detection, while the mean average precision (mAP) is a measure that we use to assess the accuracy improvements of our model.

Confusion Matrix (CM) A matrix that displays the performance of an algorithm.

Recall (R) The percentage of real positive values that are correctly identified.

Precision(P) The percentage of positive identifications that are actually correct.

Mean Average Precision (mAP) The most widely employed metric in research papers; it combines all predictions into a single value. mAP is computed by calculating the Average Precision (AP) for each class and averaging all results for all objects.

5.2 YOLOv5s

Our investigation is concentrated on the new, unlabeled data. To validate our model, we must examine how well the model performs on the labeled portion of the data while maintaining performance on the known data. In our initial trial, we utilized YOLOv5m (medium) with a batch size of 32 and 240 epochs without changing the model's hyperparameters (indicating the by default hyperparameters), and employed the pre-trained COCO weights. Even though the model has a mAp of 0.856, which indicates a promising performance throughout the training phase, Figure 23 shows that the model performs noisily in terms of learning rate, as the next paragraph explains.

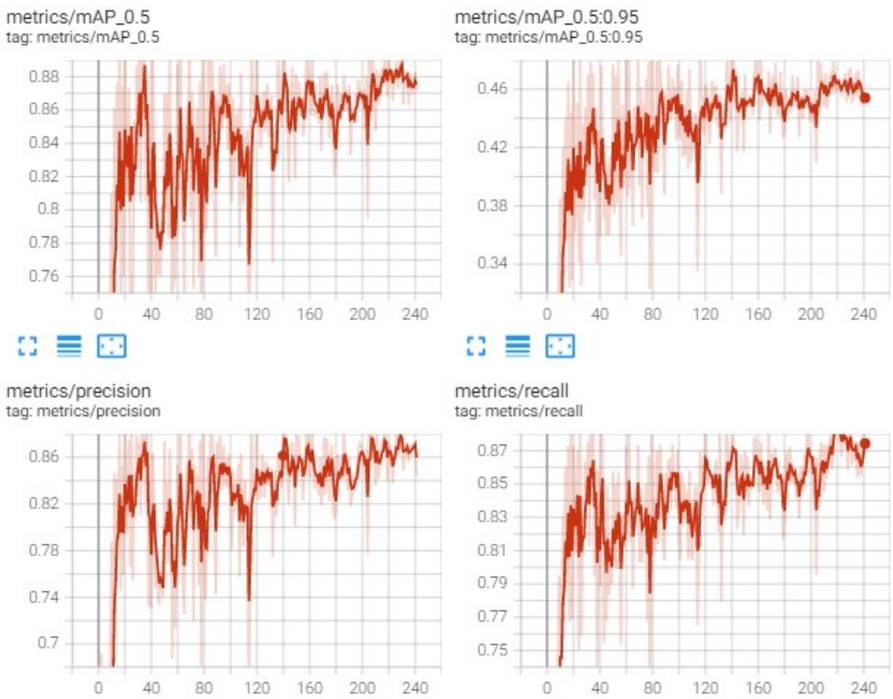

Figure 23: YOLOv5m with the default hyperparameters.

Figure 23 depicts two major issues the selected type of YOLOv5 encounters. First, the model is struggling to identify the object in an image due to the noisiness that exists in the model's mAP, precision, and recall, and second, the possibility of using the wrong type of YOLOv5. Therefore, we changed the model to the YOLOv5s (small) training model and merely increased the model's initial learning rate ($lr0$) from 0.01 to 0.001. The learning rate is a hyperparameter used to choose the step size at each iteration while aiming to minimize the loss function. Since the pre-trained weights are trained on the COCO dataset based on real-life objects, we trained YOLOv5s from scratch, indicating that the model weights are not pre-trained on any dataset (the code of training the YOLOv5s model can be found in Appendix A.2). Using no pre-trained weights, the model learned only the weight trained on the floorplan dataset. These three changes illustrate very promising results since the model performs similarly to the previous version in terms of accuracy and eliminates noise during the training process. Overall the new model performs slightly lower in terms of mAP scoring, equal to 0.833, while precision and recall scores were 0.778 and 0.824, respectively. Even if the mAP score drops slightly due to the modification of the learning rate and the selection of YOLOv5s, our model performs better during the learning process. Figure 24 demonstrates the new model's performance using 300 epochs instead of 240 that were used with YOLOv5m.

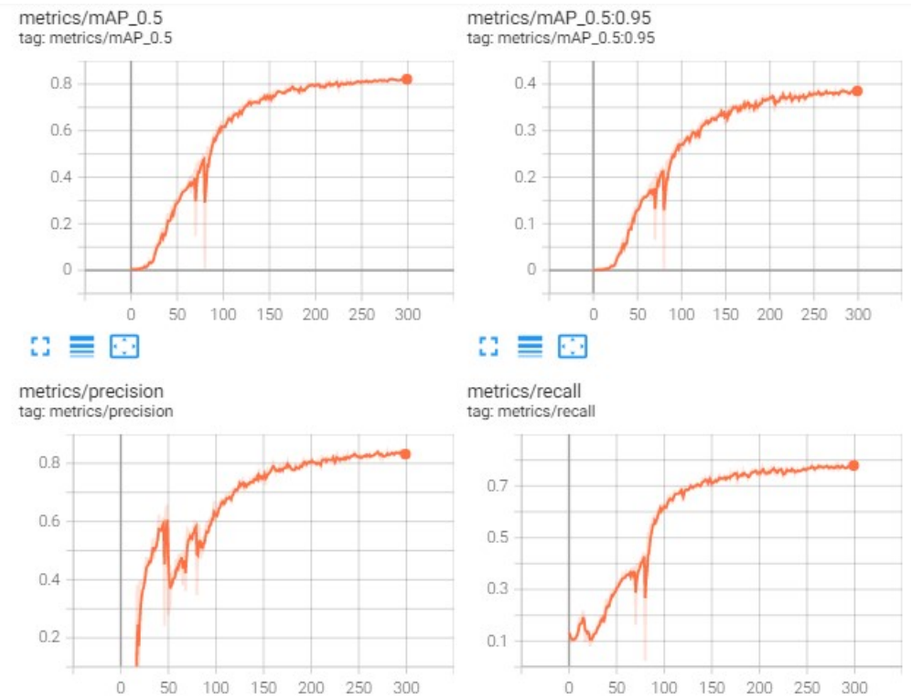

Figure 24: YOLOv5s with the modified hyperparameter and epochs=300.

YOLOv5s confusion matrix and results on each epoch are illustrated in Appendix A.4. Accordingly, Figure 29 and 30 demonstrate the overall performance of YOLOv5s on the given dataset and main evaluation metrics. Note that the illustrated results are these given by training the model with only the 400 labeled images, indicating that the model can accurately identify and classify the three distinct classes, named "High Confidence", "Medium Confidence" and "Low Confidence".

Nevertheless, Figure 25 represents the precision-recall curve. The precision-recall curve depicts the trade-off between precision and recall at various thresholds. A high area below the curve indicates both high recall and precision. Also, high accuracy is correlated with a low false positive rate, and high recall is correlated with a low false negative rate.

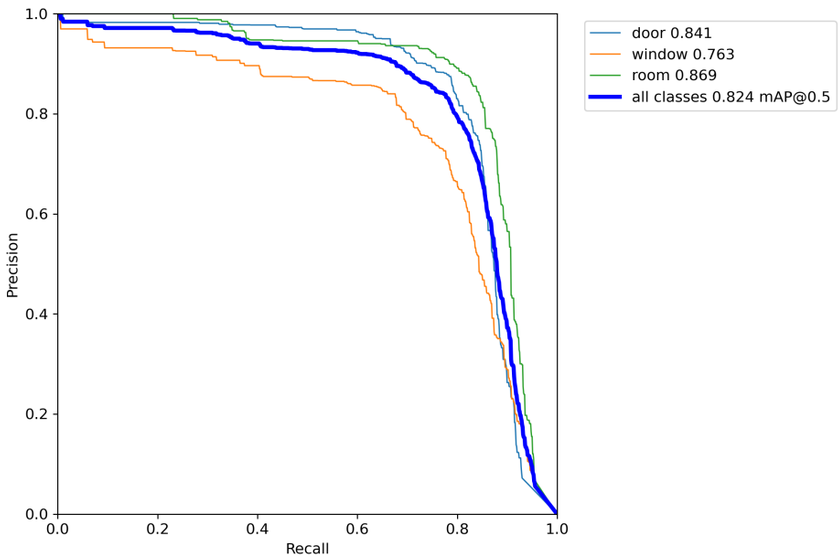

Figure 25: Precision-Recall Curve of YOLOv5s.

Our model's precision-recall pattern reveals a bowing curve toward the point $(1, 1)$, which denotes a skilled model. The combined precision-recall scores for the door, room, and window classes were

0.841, 0.869, and 0.763, respectively, giving our model a total score of 0.824.

Even if the task of our project was to annotate images with Active Learning, we detected some of the unlabeled images by setting the confidence threshold equal to 0.65 to check the image performance on unseen images. The examples in Figure 26, demonstrate that the model can detect all the classes (window, room, and door) in the given image. Even if the model is not struggling to identify the "room" class, it gets confused recognizing between the "window" class and a "door" class. As Figure 26 depicts, the model could identify only 50% of the window class (indicating identifies only one of the two windows).

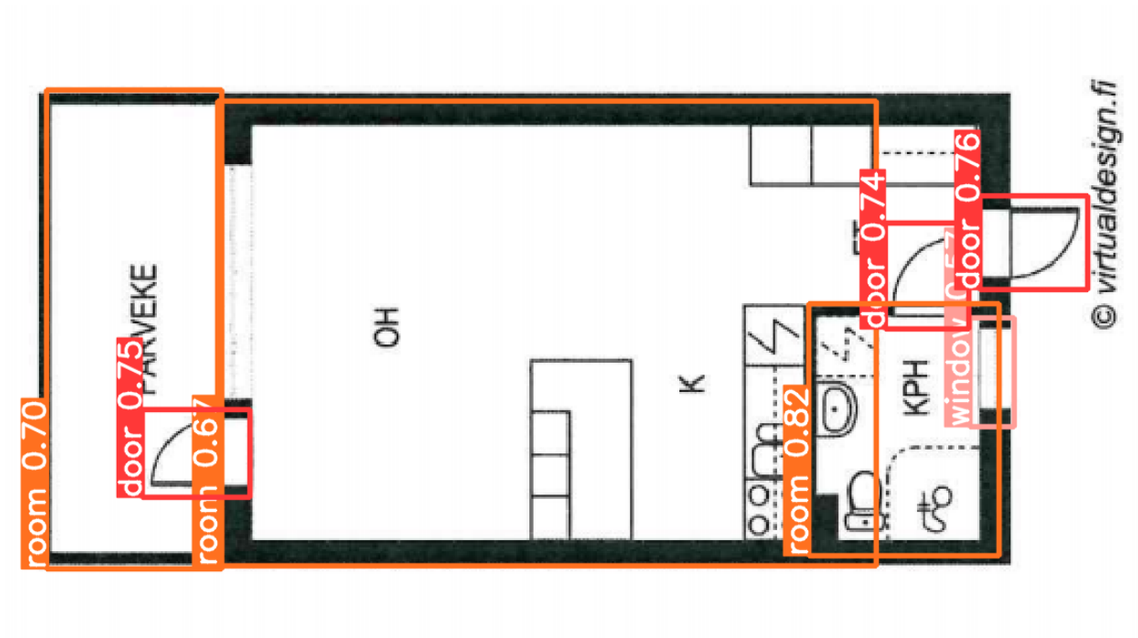

Figure 26: An image from the unlabeled dataset demonstrating the effectiveness of the model for classifying objects into the three categories of door, room, and window.

5.3 Impact of Active Learning

Our trained model provides promising results in terms of the evaluation metrics (mean Average Precision, precision, and recall). Once the Active Learning algorithm creates the three clusters, the model is re-trained with one cluster at a time; note that the clusters are divided based on the Least Confidence measure illustrating the uncertainty of the model for each cluster. Therefore, the model is re-trained with the "High Confidence" cluster and the training set; hence the model is trained with 443 images (400 training + 43 "high confidence"). Consequently, the model's performance demonstrates a slight improvement since the overall mAP of the model after the addition of the "High Confidence" cluster is 0.835. In addition to the mAP, the overall precision and recall of the model are raised to 0.839 and 0.792, respectively. The model is re-trained with 484 images, indicating the training set (400 images), the "High Confidence" cluster (43 images), and the "Mid Confidence" cluster (41 images), performing similar results as the previous training times with an mAP equals to 0.826. At the same time, the precision and recall of the model are 0.812 and 0.798, respectively. Finally, the model is re-trained with all the dataset's samples indicating the

500 images where the remaining 26 images are the "Low Confidence" images, annotated by the model. Surprisingly, the model performs an overall mAP score equal to 0.824, while the precision and recall score are 0.808 and 0.785, respectively. Table 1 reports the evaluation scores of the three clusters and the training set of the model for the three classes. Meanwhile, more detailed plots for the training of each cluster are provided in Appendix A.5, A.6, A.7.

Training set

Metrics / Classes	Precision	Recall	mAP
Door	0.857	0.79	0.841
Window	0.771	0.718	0.763
Room	0.87	0.827	0.869

High Confidence

Metrics / Classes	Precision	Recall	mAP
Door	0.827	0.807	0.861
Window	0.812	0.727	0.77
Room	0.8790.842		0.874

Mid Confidence

Metrics / Classes	Precision	Recall	mAP
Door	0.782	0.829	0.854
Window	0.801	0.725	0.763
Room	0.852	0.839	0.86

Low Confidence

Metrics / Classes	Precision	Recall	mAP
Door	0.808	0.825	0.852
Window	0.83	0.686	0.756
Room	0.823	0.845	0.864

Table 1: The Precision, Recall and mAP for each class.

Figure 27 represents the overall mAP, recall, and precision (y-axis) for the training set and the three cluster, during each epoch (x-axis). Simultaneously, Figure 27 depicts two further types of loss: the box-loss and the classification-loss. The box loss measures how accurately the algorithm can identify an object's center and how completely the anticipated bounding box encloses an object. How successfully the algorithm can determine the proper class of a given object is shown by the classification loss.

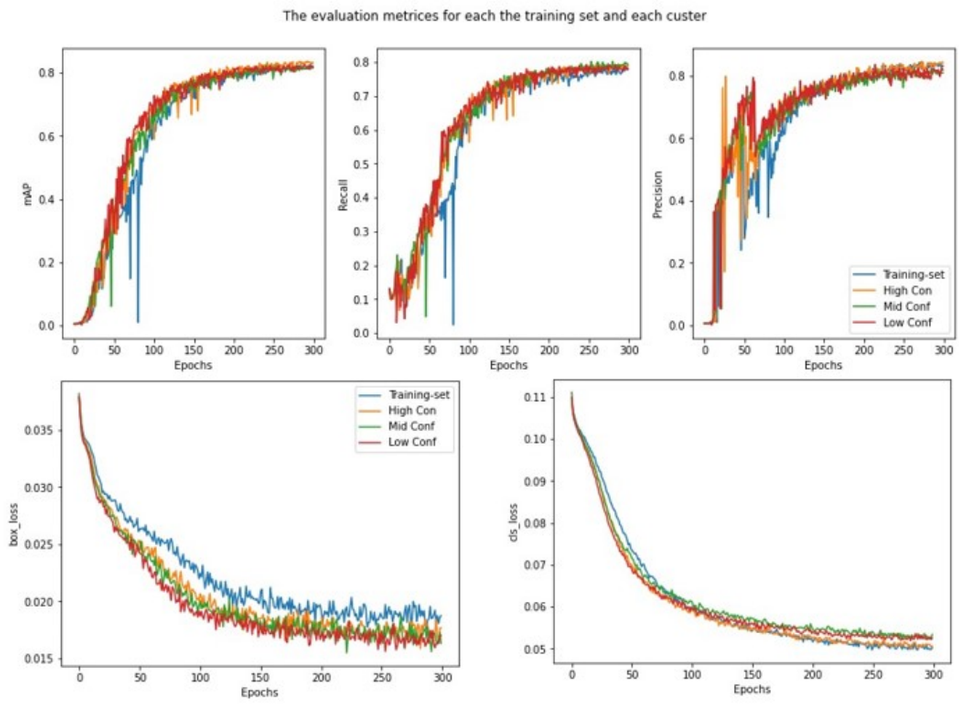

Figure 27: A summary of the model's performance during the training and the addition of clusters.

The model quickly get improved in terms of recall, mean average precision, and precision during the training of the "High Confidence" cluster; however, during the training with the "Low Confidence" cluster, the model performs similarly during the first training phase. More specifically, the model performs some ups and downs during the earliest epoch of the training phase (blue line). Nevertheless, once the model is re-trained with the created clusters, the evaluation metrics (recall, mAP, and precision) perform similarly, with the highest score of mAP ranging from 0.835 to 0.824, indicating a skilled model. Moreover, the model's box and classification loss improve during the retraining of the model with the "Low Confidence" cluster, indicating that the model may slightly drop the mAP score, but has an improvement in terms of recognizing and categorizing an object. Note that the model is trained with only 400 images at the beginning of the method and, during the Active Learning procedure, still had a high mAP score and could identify and classify objects more accurately.

Figure 28: The final trained model detecting unseen floorplan image.

Once again, we make predictions for new unseen images with the trained model on the 3 clusters. Figure 28 depicts the model's capabilities in detecting more complicated images, in comparison with Figure 26. The new unseen image includes a more complex structure of a building. Nevertheless, the model is able to identify almost every class in the floorplan image, even if the model's confidence is low compared to the first detected image. More specifically, the model is able to detect and locate the position of the windows, which illustrates the improvement of the model.

6 Conclusion

To conclude every Machine Learning model can perform much better by gathering and annotating the appropriate data. This thesis report demonstrates the capabilities of active learning using YOLOv5, a deep-learning model for object detection employed in a floorplan dataset. Furthermore, this thesis summarizes the basic concept of Artificial Neural Networks (ANNs), Convolutional Neural networks (CNNs), depicts the YOLO family, and demonstrates the main approaches of active learning while introducing the most crucial uncertainty measures. Moreover, we present a pool-based active learning approach for automated recognition and labeling of three classes of a floorplan image. The proposed method predicts/detects the labels of the objects as a standard train/test approach. Consequently, the approach clusters the predicted images based on an uncertainty measure known as Least Confidence,by creating three clusters to retrain the model. The predicted labels are used as ground-truth labels, and the model retrains itself in order to locate, recognize and classify a room, a door, and a window in the floorplan image.

6.1 Discussion & Future Work

Our approach demonstrates promising results regarding the limited dataset. Note that according to Ultralytics (creators of YOLOv5), the model needs to be trained with more than 1500 ground-truth images to have accurate results. Consequently, our model's initial training set consists of 400 images; lower than one-third of the number of images that the creators proposed. In constant to traditional pool-based approaches [(Marbinah 2021), (Nigam & McCallum 1998)], our approach uses the model as the oracle for automatically annotate the images. Even if the model was trained with limited images, the mAP of the model is not lower than 0.82. Even if the model could identify all of the classes after its re-training, the model is struggling to identify the "window" class due to the limited dataset.

Our future works mainly focuses on a comparison between different uncertainty measures when we use a larger dataset. According to these changes in our method, we aim to build an even more accurate model.

A Appendices

A.1 Data Structure

```
#mount with my dirve
from google.colab import drive
drive.mount('/content/dirve')

#list directory contents: view all files within the specified directory
!ls '/content/dirve/MyDrive/Active_Learning/Dataset'

#unzip the label file from colab / exctract evry file in my zip file
!unzip '/content/dirve/MyDrive/Active_Learning/Dataset/sub_dataset.zip' -d
    '/content/dirve/MyDrive/Active_Learning/Dataset'

#create a list with the id of each image
id = list(range(636, 1136))

#percantage of train, validation and test
train_percatnage = int(len(id) * 0.8)
val_percatnage = int(len(id)*0.2)

#Create lists for the train and validation with their class id
train_id_list = id[: train_percatnage ]
val_id_list  =id[train_percatnage : train_percatnage +  val_percatnage]

def common_data(list1, list2, train_per, val_per):
    result = False
    if len(list1) == train_per and len(list2) == val_per:
      print('The list have the same lenght')
    # traverse in the 1st list
      for x in list1:
          # traverse in the 2nd list
          for y in list2:
              # if one common
              if x == y:
                  result = True
                  print('The list does  have common data')


    return result

common_data(train_id_list, val_id_list, train_percatnage, val_percatnage)


```

```
# create the paths for the images and the txt files
import os

def images_path(path, ids):
  images = []
  for id in ids:
    im = os.path.join(path, '{}.png'.format(id))
    images.append(im)
  return images


def text_path(path, ids):
  text = []
  for id in ids:
    txt = os.path.join(path, '{}.txt'.format(id))
    text.append(txt)
  return text

path = '/content/dirve/MyDrive/Active_Learning/Dataset/sub_dataset'

train_images_path = images_path(path = path, ids = train_id_list)
train_text_path = text_path(path = path, ids = train_id_list)

val_images_path = images_path(path = path, ids = val_id_list)
val_text_path = text_path(path = path, ids = val_id_list)


#Create directories for the training and validation set
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/training_set')
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/validation_set')


#create direcories for the training images and text files
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/training_set/
    images')
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/training_set/
    labels')

#create direcories for the training images and text files
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/validation_set/
    images')
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/validation_set/
    labels')

# copy the images  and labels in the new folders
```

```
import shutil

def move_images(source, destination):
  for src in source:
    dest = shutil.move(src, destination)  #shutil.copy -> to copy the
    images

#Destination of the training images and labels
destination_train_images ='/content/dirve/MyDrive/Active_Learning/Dataset/
    training_set/images'
destination_train_labels ='/content/dirve/MyDrive/Active_Learning/Dataset/
    training_set/labels'

#Destination of the validation images and labels
destination_val_images ='/content/dirve/MyDrive/Active_Learning/Dataset/
    validation_set/images'
destination_val_labels ='/content/dirve/MyDrive/Active_Learning/Dataset/
    validation_set/labels'

move_images(source =train_images_path, destination =
    destination_train_images)
move_images(source =train_text_path, destination =
    destination_train_labels)

move_images(source = val_images_path, destination = destination_val_images
    )
move_images(source = val_text_path, destination = destination_val_labels)

clases_txt = '/content/dirve/MyDrive/Active_Learning/Dataset/sub_dataset/
    classes.txt'
destinations_paths =[destination_train_labels, destination_val_labels]

for dest in destinations_paths:
  shutil.copy(clases_txt, dest)

# Create a test file and copy the validation images with out the labels
import os
os.mkdir('/content/dirve/MyDrive/Active_Learning/Dataset/unlabeled_data')
!cp -av '/content/dirve/MyDrive/Active_Learning/Dataset/validation_set/
    images' '/content/dirve/MyDrive/Active_Learning/Dataset/unlabeled_data'
```

Listing 1: Create The data structure for YOLOv5.

A.2 YOLOv5s Training

The initial step in training any YOLOv5 model, according to the Ultralytics directory, is to clone the repository and install requirements.txt in a Python environment. Therefore, the most recent YOLOv5 release models and inbuilt datasets are immediately downloaded. As can be seen, the batch size is equal to 16, and the epochs are 300. Note that the data file name is "coco128"; however, the file paths directories had changed as we mentioned in Section 4.1. The data represent the path to the data-configurations file, while the "cfg " hyperparameter is the path to the model-configurations file. The weights hyperparameter is the path to initial weights, though the model is trained from snatch. Finally, the "cache" hyperparameter represents training the model faster, while the "hyp" represents the path to the modified hyperparameters of the model.

```

# clone the YOLOv5
!git clone https://github.com/ultralytics/yolov5
# Change the directory
%cd yolov5
# install the requirmnets
!pip install -qr requirements.txt
!python train.py --img 416 --batch 16 --epochs 300 --data coco128.yaml --
    cfg yolov5s.yaml --weights '' --hyp /content/yolov5/data/hyps/hyp.
    scratch-low.yaml --cache
```

Listing 2: The training of YOLOv5s.

A.3 Active Learning Algorithm

```

def Results_dict(path, id, model):

  '''
  Aim: sotre the redults of a spesific image in a dictionary
  Input: path of the image and the id
  output: Results of the image in dictionary format
  '''
  image_id = int(id[:4])

  #path
  unlabeld_img_path = os.path.join(path, '{}.png'.format(image_id))

  #Resutls
  results = model(unlabeld_img_path)
  results.xyxy[0] # img1 predictions (tensor)
  img_df = results.pandas().xyxy[0] # img1 predictions (pandas)

```

```
  return img_df, image_id

def Average_LC(img_df):
    '''
    Aim: find the Avarage LC for a specific image
    Input: Dataframe given by the model (df)
    Output: avg_lc (float), updated df
    '''

    #Define the vairables
    num_classes = 3

    #store the LEAST confidence in a list
    conf = [c for c in img_df.confidence]
    least_conf = [1 - lc for lc in conf]

    #LC formula
    Least_Conf_devision = [(num_classes * LCD) / (num_classes - 1) for LCD
    in least_conf]
    img_df['Least_Confidence_per_BB'] = Least_Conf_devision

    #AVERAGE LC
    avg_lc = sum(img_df.Least_Confidence_per_BB) / len(img_df.
    Least_Confidence_per_BB)

    return avg_lc, img_df


import pandas as pd

def Cluster_Average_LC(measure, measure_id):
  cluster_1_id, cluster_1_conf = [], []
  cluster_2_id, cluster_2_conf = [], []
  cluster_3_id, cluster_3_conf = [], []

  for indx, avg in enumerate(measure):
    if avg < 0.6:
      cluster_1_id.append(measure_id[indx][:4])
      cluster_1_conf.append(avg)
    elif avg >= 0.6 and avg < 0.75:
      cluster_2_id.append(measure_id[indx][:4])
      cluster_2_conf.append(avg)
    elif avg >= 0.75:
      cluster_3_id.append(measure_id[indx][:4])
      cluster_3_conf.append(avg)
```

```
62
63
64    Cluster_1 = pd.DataFrame({'ID': cluster_1_id, 'Average_LC':
       cluster_1_conf})
65    Cluster_2 = pd.DataFrame({'ID': cluster_2_id, 'Average_LC':
       cluster_2_conf})
66    Cluster_3 = pd.DataFrame({'ID': cluster_3_id, 'Average_LC':
       cluster_3_conf})
67
68    return Cluster_1, Cluster_2, Cluster_3
69
70
71  import cv2
72
73 def transform_to_yolo_format(uncertenty_id, results):
74    '''
75    Aim: transform the output from the selected images to yolo format
76    Input: ID of the image (list)
77    '''
78    yolo_format = {}
79    path = '/content/dirve/MyDrive/Active_Learning/Dataset/unlabeled_data/
       images'
80
81    for indx, id in enumerate(uncertenty_id):
82
83      img_path = os.path.join(path, '{}.png'.format(id))
84      im = cv2.imread(img_path)
85      image_h, image_w, _ = im.shape
86    #Store the vlaues in a list
87      object_class = [c for c in results[uncertenty_id[indx]]['class']]
88      xmax = [xma for xma in results[uncertenty_id[indx]].xmax]
89      xmin = [xmi for xmi in results[uncertenty_id[indx]].xmin]
90      ymax = [yma for yma in results[uncertenty_id[indx]].ymax]
91      ymin = [ymi for ymi in results[uncertenty_id[indx]].ymin]
92
93      #Transform to yolov5 format
94      height = [float(str((a - b)/image_h)[:7]) for a, b in zip(xmax, xmin)]
95      width  = [float(str((c - d)/image_w)[:7]) for c, d in zip(ymax, ymin)]
96      x_center = [float(str(((e + f)/ 2)/image_w)[:7]) for e, f in zip(xmax,
       xmin)]
97      y_center = [float(str(((g + h)/ 2)/image_h)[:7]) for g, h in zip(ymax,
       ymin)]
98
99      #Normalize the hight, width x and y co-ordinates (the images are not
       reshaped, thus every image will be normalized with their
```

```
    # corresponding measures)
    #read the image and output the width and height


    #store them as a Dataframe
    df_yolo = pd.DataFrame({'class': object_class, 'x_center':x_center, '
    y_center':y_center,
                            'width':width, 'height':height })
    yolo_format[id] = df_yolo

  return yolo_format

 #Save the annotation in the training label and the coresponding images to
     the training image and train yolov5 again
import os
import shutil

def move_and_save(new_annotations):

  path = '/content/dirve/MyDrive/Active_Learning/Dataset/unlabeled_data/
    images'
  destination = '/content/dirve/MyDrive/Active_Learning/Dataset/
    training_set/images'

  for key in new_annotations.keys():
    #Save the annotation to the training labels
    txt_annotation = new_annotations[key]
    txt_annotation.to_csv('/content/dirve/MyDrive/Active_Learning/Dataset/
    training_set/labels/{}.txt'.format(key),
                          header=None, index=None, sep= " ")

    #copy  the image to the file
    img_path = os.path.join(path, '{}.png'.format(key))
    dest = shutil.move(img_path, destination) #copy


import torch
import pandas as pd
import os

  # Model
model = torch.hub.load('ultralytics/yolov5', 'custom', path = '/content/
    yolov5/runs/train/exp/weights/best.pt' )
unlabel_images_path = '/content/dirve/MyDrive/Active_Learning/Dataset/
    unlabeled_data/images'
```

```
138
139 # Variables
140 General_avg_results, avg_LC_results = {}, {}
141
142 for id in os.listdir(unlabel_images_path):
143       # Take the results from the model as a dataframe and the
    coresponding id of the image
144       #If the i is higher than 1 then we will use the new update model
145       Images_resutls, ID = Results_dict(path = unlabel_images_path, id =
    id, model = model)
146       #Update the Results of the dataframe with a new column named
    Least_Confidence_BB (The LC of each bounding box)
147       # Avg_LC is the average Least confidence of each image
148       Avg_LC, Update_avg_results = Average_LC(Images_resutls)
149       #Store to a dectionary the image result with the coresponding ID
150       General_avg_results[str(ID)] = Update_avg_results
151       #Store the average LC for the specific image
152       avg_LC_results[str(ID)] = Avg_LC
153
154     #Create a Dataframe with the ID's of each image and the corresponding
    Average LEast Confidence
155 avg_id = [key for key in avg_LC_results.keys()]
156 avg_lc = [lc[1] for lc in avg_LC_results.items()]
157 Avg_df = pd.DataFrame({'ID':avg_id, 'Avg_LC':avg_lc})
158 Avg_Cluster1, Avg_Cluster2, Avg_Cluster3 =  Cluster_Average_LC(measure =
    Avg_df.Avg_LC, measure_id = Avg_df.ID)
159
160 for Clusters in [Avg_Cluster1, Avg_Cluster2, Avg_Cluster3]:
161   new_trained_imgs_id = [id for id in Clusters.ID]
162   # Transofrm to yolov5 annotation
163   yolo_annotations = transform_to_yolo_format(uncertenty_id =
     new_trained_imgs_id, results = General_avg_results)
164   #Move to the training directory
165   move_and_save(yolo_annotations)
166   #Train yolo with the new training set merged with the old
167   !python train.py --img 416 --batch 16 --epochs 300 --data coco128.yaml
     --cfg yolov5s.yaml --weights '' --hyp /content/yolov5/data/hyps/hyp.
     scratch-low.yaml --cache
```

Listing 3: Active Learning Algorithm.

A.4 YOLOv5s training results

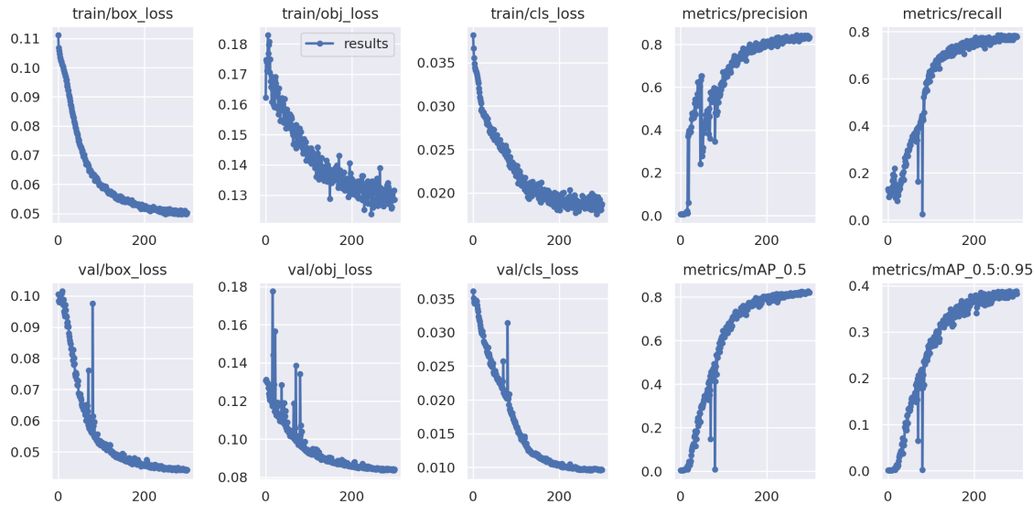

Figure 29: The results for YOLOv5s.

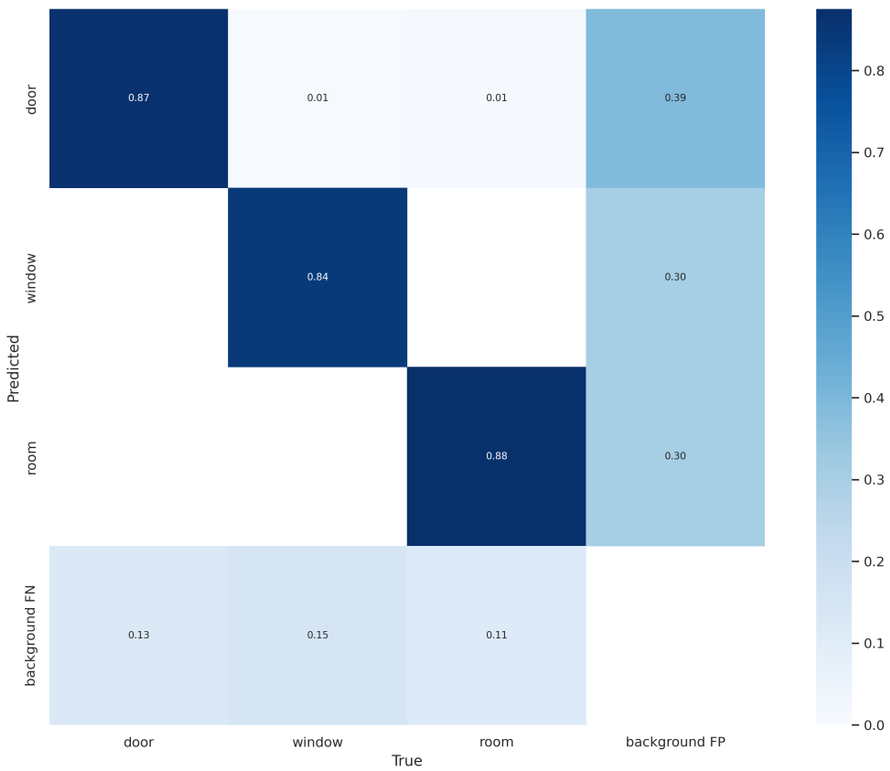

Figure 30: The confusion matrix of the training set.

A.5 "High Confidence" Cluster

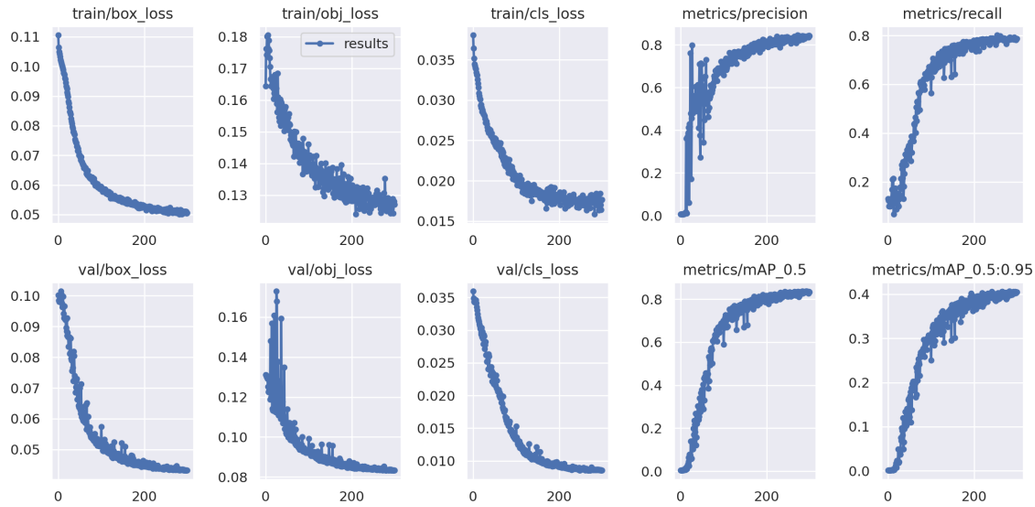

Figure 31: The results with the "High Confidence" Cluster.

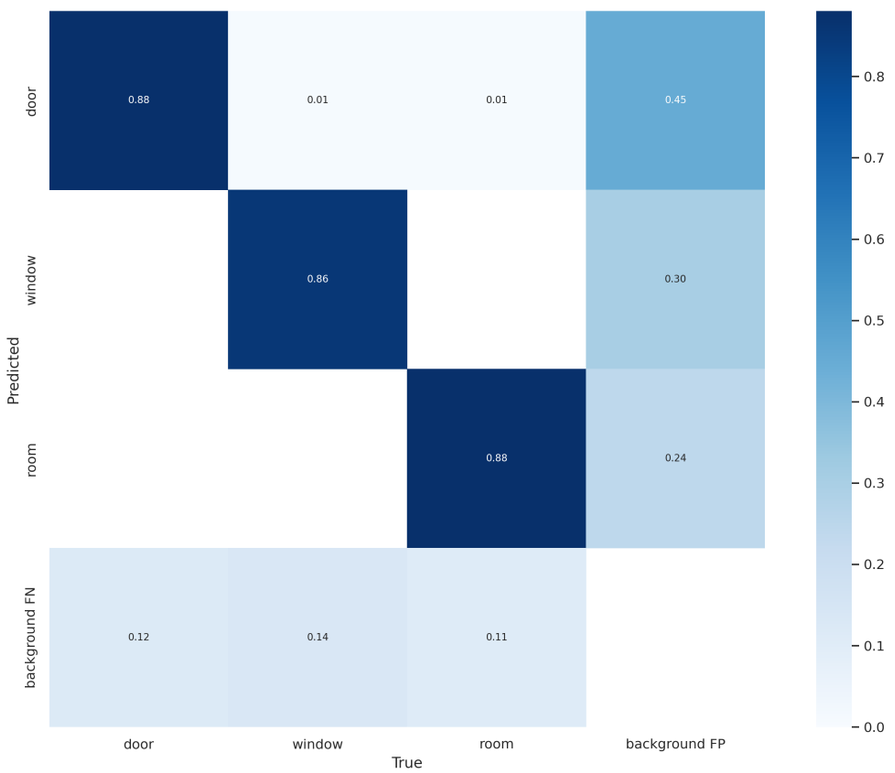

Figure 32: The confusion matrix with the "High Confidence" Cluster.

A.6 "Mid Confidence" Cluster

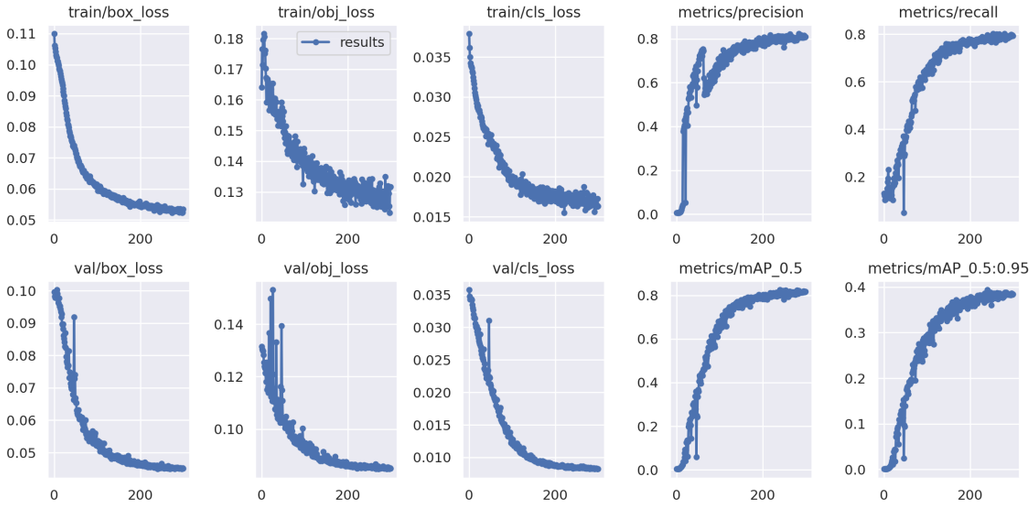

Figure 33: The results with the "Mid Confidence" Cluster.

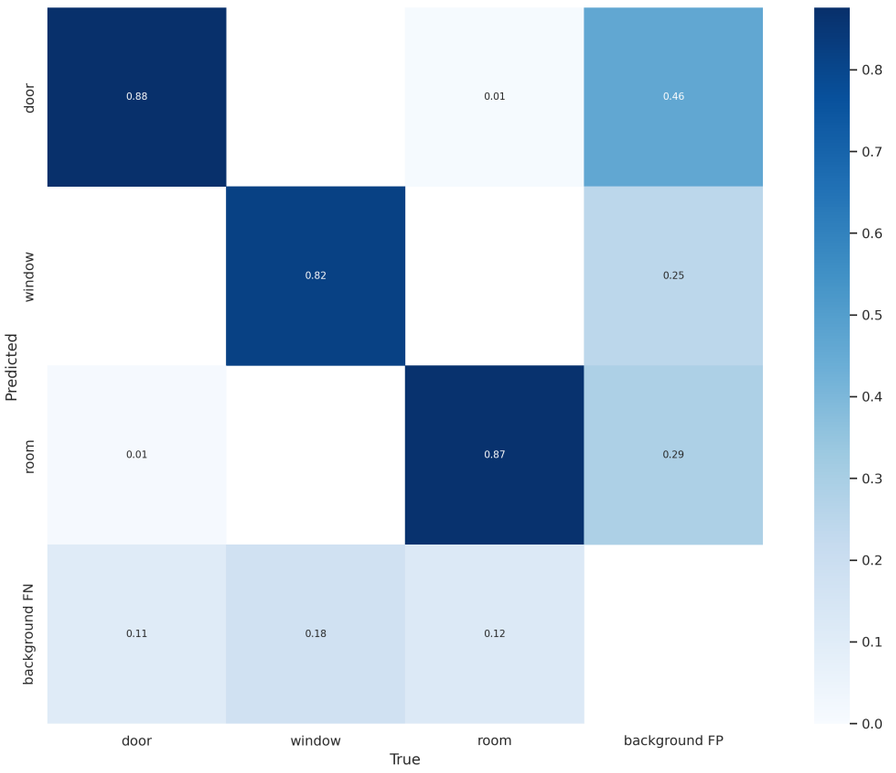

Figure 34: The confusion matrix with the "Mid Confidence" Cluster.

A.7 "Low Confidence" Cluster

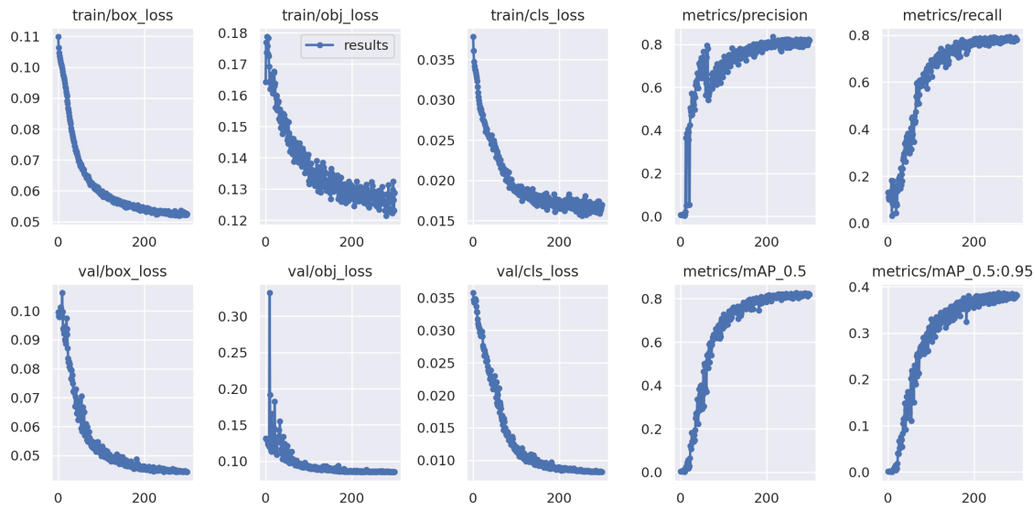

Figure 35: The results with the "Low Confidence" Cluster.

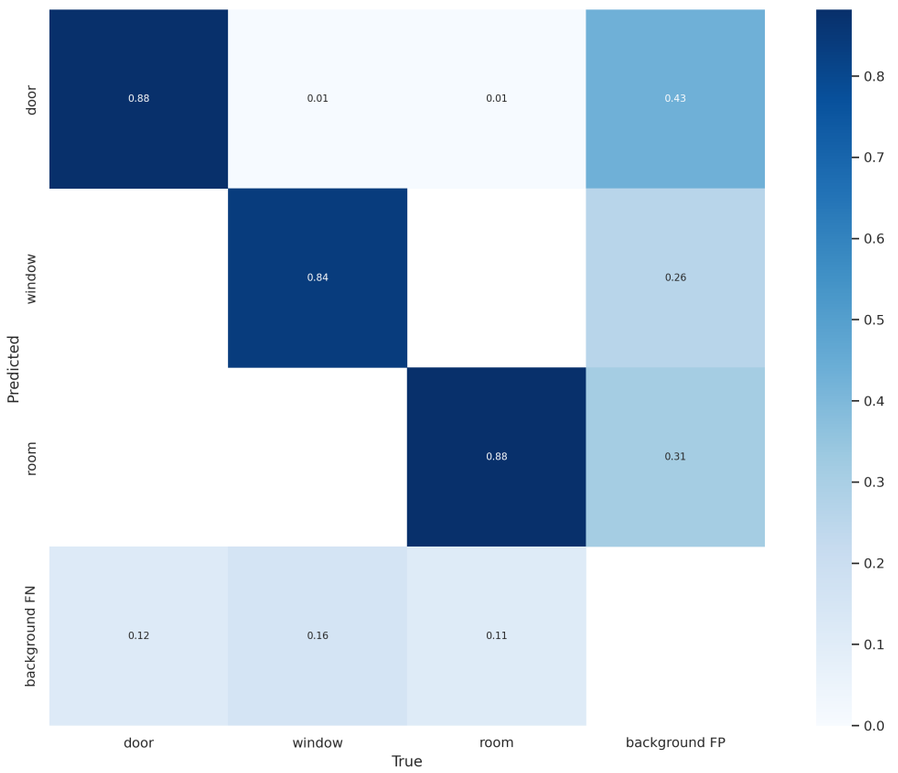

Figure 36: The confusion matrix with the "Low Confidence" Cluster.

References

Abdel-Jaber, H., Devassy, D., Al Salam, A., Hidaytallah, L. & EL-Amir, M. (2022), 'A review of deep learning algorithms and their applications in healthcare', *Algorithms* **15**(2), 71.

Adarsh, P., Rathi, P. & Kumar, M. (2020), Yolo v3-tiny: Object detection and recognition using one stage improved model, *in* '2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)', IEEE, pp. 687–694.

Agrawal, A. (2017), 'Loss functions and optimization algorithms. demystified'.
URL: *https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c*

Alto, V. (2020), 'Border effects and transition steps in convolutional neural networks'.
URL: *https://medium.com/dataseries/border-effects-and-transition-steps-in-convolutional-neural-networks-75bc6a44fd43*

Aly, G. H., Marey, M., El-Sayed, S. A. & Tolba, M. F. (2021), 'Yolo based breast masses detection and classification in full-field digital mammograms', *Computer Methods and Programs in Biomedicine* **200**, 105823.

Angluin, D. (1988), 'Queries and concept learning', *Machine learning* **2**(4), 319–342.

Ankushsharma (2020), 'Yolo v1 architecture'.
URL: *https://medium.com/@ankushsharma2805/yolo-v1-v2-v3-architecture-1ccac0f6206e: :text=YOLOV1%20uses%20features%20from%20the,boxes%20and%20their%20confidence%20scor*

Balsys, R. (2019), 'Yolov3 theory explained'.
URL: *https://pylessons.com/YOLOv3-introduction*

Bandyopadhyay, H. (2022), 'Yolo: Real-time object detection explained'.
URL: *https://www.v7labs.com/blog/yolo-object-detectionh5*

Barducci, A. & Marinai, S. (2012), Object recognition in floor plans by graphs of white connected components, *in* 'Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)', IEEE, pp. 298–301.

Benjdira, B., Khursheed, T., Koubaa, A., Ammar, A. & Ouni, K. (2019), Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3, *in* '2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)', IEEE, pp. 1–6.

Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y. M. (2020), 'Yolov4: Optimal speed and accuracy of object detection', *arXiv preprint arXiv:2004.10934* .

Bottou, L. et al. (1991), 'Stochastic gradient learning in neural networks', *Proceedings of Neuro-Nımes* **91**(8), 12.

Brownlee, J. (2017), 'Gentle introduction to the adam optimization algorithm for deep learning'.
URL: *https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/*

Brownlee, J. (2019*a*), 'A gentle introduction to batch normalization for deep neural networks'.
URL: *https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/*

Brownlee, J. (2019*b*), 'A gentle introduction to cross-entropy for machine learning'.
URL: *https://machinelearningmastery.com/cross-entropy-for-machine-learning/: :text=Cross%2Dentropy%20can%20be%20calculated,*%20log(Q(x))*

Brownlee, J. (2019*c*), 'A gentle introduction to object recognition with deep learning'.
URL: *https://machinelearningmastery.com/object-recognition-with-deep-learning/*

Brownlee, J. (2019*d*), 'A gentle introduction to pooling layers for convolutional neural networks'.
URL: *https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/: :text=A%20pooling%20layer%20is%20a,Convolutional%20Layer*

Brownlee, J. (2019*e*), 'A gentle introduction to the rectified linear unit (relu)'.
URL: *https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/*

Brownlee, J. (2019*f*), 'Loss and loss functions for training deep learning neural networks'.
URL: *https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/*

Brownlee, J. (2019*g*), 'Understand the impact of learning rate on neural network performance'.
URL: *https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/: :text=The%20learning%20rate%20controls%20how, and%20require%20fewer%20training%20epochs.*

Brownlee, J. (2021*a*), 'Difference between backpropagation and stochastic gradient descent'.
URL: *https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/: :text=Stochastic%20Gradient%20Descent%20is%20an%20optimization%20algorithm%20that*

Brownlee, J. (2021*b*), 'What is semi-supervised learning'.
URL: *https://machinelearningmastery.com/what-is-semi-supervised-learning/*

Brust, C.-A., Käding, C. & Denzler, J. (2018), 'Active learning for deep object detection', *arXiv preprint arXiv:1809.09875* .

Calle (2020*a*), 'Mean squared error'.
URL: *https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error*

Calle (2020*b*), 'Stochastic gradient descent'.

URL: *https://peltarion.com/knowledge-center/documentation/modeling-view/run-a-model/optimizers/stochastic-gradient-descent*

Chakure, A. (2021), 'All you need to know about yolo v3 (you only look once)'.

URL: *https://dev.to/afrozchakure/all-you-need-to-know-about-yolo-v3-you-only-look-once-e4m*

Chen, Z.-H. & Juang, J.-C. (2022), 'Yolov4 object detection model for nondestructive radiographic testing in aviation maintenance tasks', *AIAA journal* **60**(1), 526–531.

Ching (2021), 'Object detection explained: Yolo v2'.

URL: *https://medium.com/mlearning-ai/object-detection-explained-yolo-v2-3e3086789ffb*

Chitale, P. A., Kekre, K. Y., Shenai, H. R., Karani, R. & Gala, J. P. (2020), Pothole detection and dimension estimation system using deep learning (yolo) and image processing, *in* '2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)', IEEE, pp. 1–6.

Choudhary, S. (2021), 'Yolov4: A quick overview'.

URL: *https://medium.com/analytics-vidhya/yolov4-a-quick-overview-d98e29c22771*

Christiansen, A. (2018), 'Anchor boxes — the key to quality object detection'.

URL: *https://towardsdatascience.com/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9*

Cohen, J. (2020), 'Introduction to yolov4: Research review'.

URL: *https://www.thinkautonomous.ai/blog/introduction-to-yolov4-research-review/*

Cohen, O. (2018), 'Active learning tutorial'.

URL: *https://towardsdatascience.com/active-learning-tutorial-57c3398e34d*

Cohn, D. (2010), *Active Learning*, Springer US, Boston, MA, pp. 10–14.

URL: *https://doi.org/10.1007/978-0-387-30164-8*$_6$

Cormack, G. V. & Grossman, M. R. (2014), Evaluation of machine-learning protocols for technology-assisted review in electronic discovery, *in* 'Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval', pp. 153–162.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H. & Wei, Y. (2017), Deformable convolutional networks, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 764–773.

Dave, H. (2020), 'Active learning sampling strategies'.

URL: *https://medium.com/@hardik.dave/active-learning-sampling-strategies-f8d8ac7037c8: :text=Stream%2Dbased%20Selective%20Sampling%3A%20In,based%20on%20a%20query%20strat*

Demush, R. (2019), 'A brief history of computer vision (and convolutional neural networks)'.

URL: *https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3*

Deng, J., Xuan, X., Wang, W., Li, Z., Yao, H. & Wang, Z. (2020), A review of research on object detection based on deep learning, *in* 'Journal of Physics: Conference Series', Vol. 1684, IOP Publishing, p. 012028.

Desai, S. V. & Balasubramanian, V. N. (2020), Towards fine-grained sampling for active learning in object detection, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops', pp. 924–925.

Dickson, B. (2021), 'An introduction to object detection with deep learning'.
URL: *https://bdtechtalks.com/2021/06/21/object-detection-deep-learning/*

Du, J. (2018), Understanding of object detection based on cnn family and yolo, *in* 'Journal of Physics: Conference Series', Vol. 1004, IOP Publishing, p. 012029.

Du, L., Zhang, R. & Wang, X. (2020), Overview of two-stage object detection algorithms, *in* 'Journal of Physics: Conference Series', Vol. 1544, IOP Publishing, p. 012033.

Dumoulin, V. & Visin, F. (2016), 'A guide to convolution arithmetic for deep learning', *arXiv preprint arXiv:1603.07285* .

Feng, V. (2017), 'An overview of resnet and its variants', *Towards data science* **2**.

Ganesh, S. (2020), 'What's the role of weights and bias in a neural network?'.
URL: *https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f*

Gissin, D. & Shalev-Shwartz, S. (2019), 'Discriminative active learning', *arXiv preprint arXiv:1907.06347* .

Goyal, S., Chattopadhyay, C. & Bhatnagar, G. (2021), 'Knowledge-driven description synthesis for floor plan interpretation', *International Journal on Document Analysis and Recognition (IJDAR)* **24**(1), 19–32.

Gupta, A. (2021), 'A comprehensive guide on deep learning optimizers'.
URL: *https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/: :text=While%20training%20the%20deep%20learning,loss%20and%20improve%20the%20accuracy.*

Han, W., Coutinho, E., Ruan, H., Li, H., Schuller, B., Yu, X. & Zhu, X. (2016), 'Semi-supervised active learning for sound classification in hybrid learning environments', *PloS one* **11**(9), e0162075.

Han, X., Chang, J. & Wang, K. (2021), 'Real-time object detection based on yolo-v2 for tiny vehicle object', *Procedia Computer Science* **183**, 61–72.

HANSEN, C. (2019), 'Neural networks: Feedforward and backpropagation explained optimization'.
URL: *https://mlfromscratch.com/neural-networks-explained//*

He, K., Zhang, X., Ren, S. & Sun, J. (2015), 'Spatial pyramid pooling in deep convolutional networks for visual recognition', *IEEE transactions on pattern analysis and machine intelligence* **37**(9), 1904–1916.

Holub, A., Perona, P. & Burl, M. C. (2008), Entropy-based active learning for object recognition, *in* '2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops', IEEE, pp. 1–8.

Huang, S.-J., Jin, R. & Zhou, Z.-H. (2010), 'Active learning by querying informative and representative examples', *Advances in neural information processing systems* **23**.

Hui, J. (2018*a*), 'Real-time object detection with yolo, yolov2 and now yolov3'.
URL: *https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088*

Hui, J. (2018*b*), 'Understanding feature pyramid networks for object detection (fpn)'.
URL: *https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c*

jenn (2021), 'Continuity and differentiability'.
URL: *https://calcworkshop.com/derivatives/continuity-and-differentiability/chapter*

Jiang, P., Ergu, D., Liu, F., Cai, Y. & Ma, B. (2022), 'A review of yolo algorithm developments', *Procedia Computer Science* **199**, 1066–1073.

Jocher, G. (2020), 'ultralytics/yolov5'.
URL: *https://github.com/ultralytics/yolov5*

John, A. C. (2022), 'Active learning: Strategies, tools, and real-world use cases'.
URL: *https://neptune.ai/blog/active-learning-strategies-tools-use-cases*

Joshi, A. J., Porikli, F. & Papanikolopoulos, N. (2009), Multi-class active learning for image classification, *in* '2009 ieee conference on computer vision and pattern recognition', IEEE, pp. 2372–2379.

K, S. (2019), 'Non-maximum suppression (nms)'.

Kalervo, A., Ylioinas, J., Häikiö, M., Karhu, A. & Kannala, J. (2019), Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis, *in* 'Scandinavian Conference on Image Analysis', Springer, pp. 28–40.

Kamal, A. (2019), 'Yolo, yolov2, and yolov3: All you want to know'.
URL: *https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899*

Kapil, D. (2018*a*), 'Yolo v1 : Part 1'.
URL: *https://medium.com/adventures-with-deep-learning/yolo-v1-part-1-cfb47135f81f*

Kapil, D. (2018*b*), 'Yolo v1: Part 2'.

URL: *https://medium.com/@divakar*$_2$39/$yolo - v1 - part - 2 - bfc686ae5560$

Karimi, G. (2021), 'Introduction to yolo algorithm for object detection'.

URL: *https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/*

Kathuria, A. (2018), 'What's new in yolo v3?'.

URL: *https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b*

Kezhuang, E. (2020), 'Notes on object detection: Yolo9000(v2)'.

URL: *https://zhuanlan.zhihu.com/p/208452160*

Kibria, S. B. & Hasan, M. S. (2017), An analysis of feature extraction and classification algorithms for dangerous object detection, *in* '2017 2nd International Conference on Electrical & Electronic Engineering (ICEEE)', IEEE, pp. 1–4.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kumar, P. & Gupta, A. (2020), 'Active learning query strategies for classification, regression, and clustering: a survey', *Journal of Computer Science and Technology* **35**(4), 913–945.

Kwiatkowski, R. (2021), 'Gradient descent algorithm — a deep dive'.

URL: *https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21*

Li, J., Ge, H., Zhang, Z., Wang, W. & Yang, Y. (n.d.), 'Traffic multiple target detection on yolov2'.

Li, Y., Dua, A. & Ren, F. (2020), Light-weight retinanet for object detection on edge devices, *in* '2020 IEEE 6th World Forum on Internet of Things (WF-IoT)', IEEE, pp. 1–6.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. & Belongie, S. (2017), Feature pyramid networks for object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 2117–2125.

Liu, S., Qi, L., Qin, H., Shi, J. & Jia, J. (2018), Path aggregation network for instance segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 8759–8768.

Liu, Y. (2004), 'Active learning with support vector machine applied to gene expression data for cancer classification', *Journal of chemical information and computer sciences* **44**(6), 1936–1941.

Loyola-Gonzalez, O. (2019), 'Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view', *IEEE Access* **7**, 154096–154113.

Lu, X., Ji, J., Xing, Z. & Miao, Q. (2021), 'Attention and feature fusion ssd for remote sensing object detection', *IEEE Transactions on Instrumentation and Measurement* **70**, 1–9.

Malik, F. (2019*a*), 'Neural networks bias and weights'.

URL: *https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da*

Malik, F. (2019*b*), 'What are hidden layers?'.

URL: *https://medium.com/fintechexplained/what-are-hidden-layers-4f54f7328263*

Mantripragada, M. (2020), 'Digging deep into yolo v3 - a hands-on guide part 1'.

URL: *https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29*

Marbinah, J. (2021), 'Hybrid pool based deep active learning for object detection using intermediate network embeddings'.

McGonagle. J, García J. A, M. S. (2022), 'Feedforward neural networks'.

URL: *https://brilliant.org/wiki/feedforward-neural-networks/*

McGrath, J. (2022), 'Active learning: an overview'.

URL: *https://snorkel.ai/active-learning/*

Minty, G. J. (1964), 'On the monotonicity of the gradient of a convex function.', *Pacific Journal of Mathematics* **14**(1), 243–247.

Mishra, M. (2020), 'Convolutional neural networks, explained'.

URL: *https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939: :text=A%20Convolutional%20Neural%20Network%2C%20also,binary%20representation%20of%2*

Mishra, S., Hashmi, K. A., Pagani, A., Liwicki, M., Stricker, D. & Afzal, M. Z. (2021), 'Towards robust object detection in floor plan images: A data augmentation approach', *Applied Sciences* **11**(23), 11174.

MK, G. (2020), 'Basic cnn architecture: Explaining 5 layers of convolutional neural network'.

URL: *https://www.upgrad.com/blog/basic-cnn-architecture/*

Mohan, A. (2020*a*), 'Review on yolov1'.

URL: *https://medium.datadriveninvestor.com/review-on-yolov1-3c85304b617d*

Mohan, A. (2020*b*), 'Review on yolov2'.

URL: *https://medium.datadriveninvestor.com/review-on-yolov2-11e93c5ea3f1*

Mostafa, S. & Wu, F.-X. (2021), Diagnosis of autism spectrum disorder with convolutional autoencoder and structural mri images, *in* 'Neural Engineering Techniques for Autism Spectrum Disorder', Elsevier, pp. 23–38.

Nakahara, H., Fujii, T. & Sato, S. (2017), A fully connected layer elimination for a binarizec convolutional neural network on an fpga, *in* '2017 27th international conference on field programmable logic and applications (FPL)', IEEE, pp. 1–4.

Nelson, J. (2020), 'What is active learning?'.
URL: *https://blog.roboflow.com/what-is-active-learning/*

Nigam, K. & McCallum, A. (1998), Pool-based active learning for text classification, *in* 'Conference on Automated Learning and Discovery (CONALD)', Citeseer.

Oommen, P. (2020), 'Resnets — residual blocks deep residual learning'.
URL: *https://towardsdatascience.com/resnets-residual-blocks-deep-residual-learning-a231a0ee73d2: :text=A%20residual%20block%20is%20a,layer%20in%20the%20main%20path.*

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D. & Walsh, J. (2019), Deep learning vs. traditional computer vision, *in* 'Science and information conference', Springer, pp. 128–144.

Peixeiro, M. (2019), 'The complete guide to unsupervised learning'.
URL: *https://towardsdatascience.com/the-complete-guide-to-unsupervised-learning-ecf8b676f2af: :text=Unsupervised%20learning%20is%20a%20set,associated%20responses%20to%20each%20obser*

Prendki, J. (2020), 'Introduction to active learning'.
URL: *https://www.kdnuggets.com/2018/10/introduction-active-learning.html*

R, M. (2020), 'Panet: Path aggregation network in yolov4'.
URL: *https://medium.com/clique-org/panet-path-aggregation-network-in-yolov4-b1a6dd09d158*

Raghunandan, A., Raghav, P., Aradhya, H. R. et al. (2018), Object detection algorithms for video surveillance applications, *in* '2018 International Conference on Communication and Signal Processing (ICCSP)', IEEE, pp. 0563–0568.

Ravi, R. (n.d.), 'Mobile phones: An image classification problem a background and introduction, using supervised-learning algorithms'.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Redmon, J. & Farhadi, A. (2017), Yolo9000: better, faster, stronger, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 7263–7271.

Redmon, J. & Farhadi, A. (2018), 'Yolov3: An incremental improvement', *arXiv preprint arXiv:1804.02767* .

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X. & Wang, X. (2021), 'A survey of deep active learning', *ACM computing surveys (CSUR)* **54**(9), 1–40.

Rosebrock, A. (2016), 'Intersection over union (iou) for object detection'.
URL: *https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/*

Rothe, R., Guillaumin, M. & Gool, L. V. (2014), Non-maximum suppression for object detection by passing messages between windows, *in* 'Asian conference on computer vision', Springer, pp. 290–306.

Roy, S., Unmesh, A. & Namboodiri, V. P. (2018), Deep active learning for object detection., *in* 'BMVC', p. 91.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Saha, S. (2018), 'A comprehensive guide to convolutional neural networks — the eli5 way'.
URL: *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*

Sandelin, F. (2019), 'Semantic and instance segmentation of room features in floor plans using mask r-cnn'.

Sandhu, J. S. (2020), 'Introduction to residual networks'.
URL: *https://www.geeksforgeeks.org/introduction-to-residual-networks/*

Sanghvirajit (2021), 'A complete guide to adam and rmsprop optimizer'.
URL: *https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be*

Saxena, S. (2021*a*), 'Introduction to batch normalization'.
URL: *https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/*

Saxena, S. (2021*b*), 'Introduction to softmax for neural network'.
URL: *https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/: :text=The%20Softmax%20activation%20function%20calculates,determine%20the%20final%20probabil*

Schumann, R. & Rehbein, I. (2019), Active learning via membership query synthesis for semi-supervised sentence classification, *in* 'Proceedings of the 23rd conference on computational natural language learning (CoNLL)', pp. 472–481.

Serengil, S. I. (2017), 'Step function as a neural network activation function'.
URL: *https://sefiks.com/2017/05/15/step-function-as-a-neural-network-activation-function/: :text=Every%20logic%20function%20can%20be,AND%20gate%20or%20OR%20gate.*

Setiawan, W. & Purnama, A. (2020), Tobacco leaf images clustering using darknet19 and k-means, *in* '2020 6th Information Technology International Seminar (ITIS)', IEEE, pp. 269–273.

Settles, B. (2009), Active learning literature survey, Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Settles, B. & Craven, M. (2008), An analysis of active learning strategies for sequence labeling tasks, *in* 'proceedings of the 2008 conference on empirical methods in natural language processing', pp. 1070–1079.

Shah, D. (2020*a*), 'Yolov4 — version 1: Bag of freebies'.

URL: *https://medium.com/visionwizard/yolov4-bag-of-freebies-dc126623fc2d*

Shah, P. (2020*b*), 'Let your model select which data to learn from — basics of active learning'.

URL: *https://towardsdatascience.com/let-your-model-select-which-data-to-learn-from-basics-of-active-learning-7c56846a6c5d*

Shahid (2019), 'Convolutional neural network'.

URL: *https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529: :text=Stride%20denotes%20how%20many%20steps,By%20default%20it%20is%20one.text=We%2*

Sharma, A. (2022*a*), 'Achieving optimal speed and accuracy in object detection (yolov4)'.

URL: *https://pyimagesearch.com/2022/05/16/achieving-optimal-speed-and-accuracy-in-object-detection-yolov4/*

Sharma, A. (2022*b*), 'A better, faster, and stronger object detector (yolov2)'.

URL: *https://pyimagesearch.com/2022/04/18/a-better-faster-and-stronger-object-detector-yolov2/*

Sharma, A. (2022*c*), 'An incremental improvement with darknet-53 and multi-scale predictions (yolov3)'.

URL: *https://pyimagesearch.com/2022/05/09/an-incremental-improvement-with-darknet-53-and-multi-scale-predictions-yolov3/*

Sharma, A. (2022*d*), 'Introduction to the yolo family'.

URL: *https://pyimagesearch.com/2022/04/04/introduction-to-the-yolo-family/*

SHARMA, S. (2017), 'Activation functions in neural networks'.

URL: *https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6*

Singla, S. (2020), 'Why is batch normalization useful in deep neural networks?'.

URL: *https://towardsdatascience.com/batch-normalisation-in-deep-neural-network-ce65dd9e8dbf: :text=Batch%20normalization%20is%20a%20technique,to%20train%20deep%20neural%20networ*

Solaguren-Beascoa, A. (2020), 'Active learning in machine learning'.

URL: *https://towardsdatascience.com/active-learning-in-machine-learning-525e61be16e5*

Solawetz, J. (2020), 'Yolov4 explained'.

URL: *https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/*

Subramanyam, V. S. (2021), 'Iou (intersection over union)'.

URL: *https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef*

Sun, L.-L. & Wang, X.-Z. (2010), A survey on active learning strategy, *in* '2010 International Conference on Machine Learning and Cybernetics', Vol. 1, IEEE, pp. 161–166.

Szegedy, C., Toshev, A. & Erhan, D. (2013), 'Deep neural networks for object detection', *Advances in neural information processing systems* **26**.

Thomas, C. (2019), 'An introduction to convolutional neural networks'.
URL: *https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd*

Tiwari, R., Mane, V. & Chaugule, R. (2022), 'Analysis of object detection algorithms'.

Tran, C. V., Nguyen, T. T., Hoang, D. T., Hwang, D. & Nguyen, N. T. (2017), Active learning-based approach for named entity recognition on short text streams, *in* 'Multimedia and Network Information Systems', Springer, pp. 321–330.

Trivedi, S. (2020*a*), 'Yolov4 — version 0: Introduction'.
URL: *https://medium.com/visionwizard/yolov4-version-0-introduction-90514b413ccf*

Trivedi, S. (2020*b*), 'Yolov4 — version 0: Introduction'.
URL: *https://medium.com/visionwizard/yolov4-version-0-introduction-90514b413ccf*

Trivedi, S. (2020*c*), 'Yolov4 — version 2: Bag of specials'.
URL: *https://medium.com/visionwizard/yolov4-version-2-bag-of-specials-fab1032b7fa0*

Tsang, S.-H. (2018*a*), 'Review: Yolov1 — you only look once (object detection)'.
URL: *https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89*

Tsang, S.-H. (2018*b*), 'Review: Yolov1 — you only look once (object detection)'.
URL: *https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89*

Tsang, S.-H. (2018*c*), 'Review: Yolov2 yolo9000 — you only look once (object detection)'.
URL: *https://towardsdatascience.com/review-yolov2-yolo9000-you-only-look-once-object-detection-7883d2b02a65: :text=YOLOv2%20removes%20all%20fully%20connected,obtained%2C%20i.e.%2032%*

Tsang, S.-H. (2021), 'Review — cspnet: A new backbone that can enhance learning capability of cnn'.
URL: *https://sh-tsang.medium.com/review-cspnet-a-new-backbone-that-can-enhance-learning-capability-of-cnn-da7ca51524bf*

Walczak, S. & Cerpa, N. (1999), 'Heuristic principles for the design of artificial neural networks', *Information and software technology* **41**(2), 107–117.

Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W. & Yeh, I.-H. (2020), Cspnet: A new backbone that can enhance learning capability of cnn, *in* 'Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops', pp. 390–391.

Weng, L. (2018), 'Object detection part 4: Fast detection models'.
URL: *https://lilianweng.github.io/posts/2018-12-27-object-recognition-part-4/: :text=Direct%20location%20prediction%3A%20YOLOv2%20formulates,model%20training%20could%20becor*

Wu, D. (2018), 'Pool-based sequential active learning for regression', *IEEE transactions on neural networks and learning systems* **30**(5), 1348–1359.

Yamashita, R., Nishio, M., Do, R. K. G. & Togashi, K. (2018), 'Convolutional neural networks: an overview and application in radiology', *Insights into imaging* **9**(4), 611–629.

Yang, L., MacEachren, A. M., Mitra, P. & Onorati, T. (2018), 'Visually-enabled active deep learning for (geo) text and image classification: a review', *ISPRS International Journal of Geo-Information* **7**(2), 65.

Zhai, S., Shang, D., Wang, S. & Dong, S. (2020), 'Df-ssd: An improved ssd object detection algorithm based on densenet and feature fusion', *IEEE access* **8**, 24344–24357.

Zhang, Z. (2020), 'Yolo 2 explained'.
URL: *https://towardsdatascience.com/yolo2-walkthrough-with-examples-e40452ca265f*

Zhao, L. & Li, S. (2020), 'Object detection algorithm based on improved yolov3', *Electronics* **9**(3), 537.

Zhao, Z.-Q., Zheng, P., Xu, S.-t. & Wu, X. (2019), 'Object detection with deep learning: A review', *IEEE transactions on neural networks and learning systems* **30**(11), 3212–3232.

Ziran, Z. & Marinai, S. (2018), Object detection in floor plan images, *in* 'IAPR workshop on artificial neural networks in pattern recognition', Springer, pp. 383–394.