hYOLO Model: Enhancing Object Classification with Hierarchical Context in *YOLOv8*

Veska Tsenkova[1]* iD, Peter Stanchev[1]† iD,
Daniel Petrov[1]† iD, Deyan Lazarov[1]† iD

[1]DS and AI Solutions, Soft2RUN, Sofia, 1000, Bulgaria

Corresponding author: vtsenkova@soft2run.com

Contributing authors: pstanchev@soft2run.com; hyolopermission@soft2run.com; dlazarov@soft2run.com
†These authors contributed equally to this work.

Abstract: Current convolution neural network (CNN) classification methods are predominantly focused on flat classification which aims solely to identify a specified object within an image. However, real-world objects often possess a natural hierarchical organization that can significantly help classification tasks. Capturing the presence of relations between objects enables better contextual understanding as well as control over the severity of mistakes. Considering these aspects, this paper proposes an end-to-end hierarchical model for image detection and classification built upon the *YOLO* model family. A novel hierarchical architecture, a modified loss function, and a performance metric tailored to the hierarchical nature of the model are introduced. The proposed model is trained and evaluated on two different hierarchical categorizations of the same dataset: a systematic categorization that disregards visual similarities between objects and a categorization accounting for common visual characteristics across classes. The results illustrate how the suggested methodology addresses the inherent hierarchical structure present in real-world objects, which conventional flat classification algorithms often overlook.

Keywords: Computer vision, Hierarchical classification, Loss function, *YOLO* model

1 Introduction

Many real-world classification problems are structured hierarchically, with target classes organized into multiple levels of abstraction. In such cases, a hierarchical classification model aims not only to predict the correct class but also to do so within

1

a taxonomy that reflects the relationships among classes. For example, in biological datasets, organisms are classified into a hierarchical structure consisting of kingdom, phylum, class, order, family, genus, and species. A **flat** classification model directly classifies all the species without considering the hierarchical relationships, bypassing the broader categories such as kingdom, phylum, or class. In contrast, a **hierarchical** classification model begins by predicting the broadest category, the kingdom, and progressively refines its predictions as it moves down the taxonomy, ultimately reaching the most specific level, the species. This approach ensures that predictions are made within the context of related classes, thereby capturing the hierarchical structure and enabling the model to better manage complex relationships between different levels. In addition, hierarchical classification models offer several advantages over flat models, particularly in domains like computer vision, where large number of classes and hierarchical levels need to be processed:

Error control and More Efficient Learning:

By teaching the network to maintain predicted classes within the same hierarchical category, hierarchical models not only learn how to accurately classify objects but also how to make errors that are less harmful. This feature is particularly valuable in high-risk domains such as medical image diagnosis, where the consequences of misclassification can be significant. Rather than treating each diagnosis as an independent class, the model may first classify a skin lesion into broad risk groups such as "benign," "pre-malignant," or "malignant," before refining the prediction to a specific condition. This structure reduces the likelihood of critical errors, such as misclassifying a malignant tumor as benign, by keeping misclassifications within the same risk category. A representative example is the work by Yu et al. (2022), who proposed a class-hierarchy regularized hyperbolic embedding model for skin lesion recognition. Their results show that incorporating hierarchical structure improves both accuracy and diagnostic safety by aligning misclassifications with clinically similar categories.

In addition, instead of learning all classes simultaneously, their hierarchical model progresses through multiple levels of abstraction, beginning with broad, coarse-grained categories and incrementally refining its predictions toward more specific target classes. The model initially groups lesions into "benign," "pre-malignant," or "malignant" categories, and then further differentiates within each group (e.g., distinguishing between "basal cell carcinoma" and "squamous cell carcinoma" within the malignant category). This stepwise, sequential learning process enhances the model's convergence rate and improves generalization, as it focuses on high-level features before addressing more complex details.

Reduced Complexity: In complex classification tasks with a large number of classes, flat classification models may struggle to learn simultaneously discriminative features for each class. Hierarchical models manage complexity by breaking down classification tasks into manageable subproblems. The model initially deals with simpler, broader classes, and gradually adds more detailed predictions as the hierarchy deepens. This reduces the difficulty of learning discriminative features for every class at once, making the task more manageable. For example, a hierarchical classification model diagnosing lung conditions (Yang et al. (2020)) may first classify images into broader categories, such as "healthy" or "abnormal." Within the "abnormal" category,

the model could further differentiate between conditions like "pneumonia," "tuberculosis," or "lung cancer." This structured approach allows the model to focus on learning general patterns for healthy vs. abnormal tissue, and then focus on the more challenging task of distinguishing between various diseases within the abnormal group.

Improved Interpretability and Scalability:

Hierarchical models naturally capture semantic relationships between classes, enhancing both interpretability and scalability: two critical features for domains such as healthcare or inventory management. In terms of interpretability, these models enable users to trace predictions through successive layers of abstraction, offering insight into how decisions are made. For example, in chest X-ray analysis (Chen, Miao, Xu, Hager, and Harrison (2020)), a model may move from broad labels like "normal" or "abnormal" to increasingly specific diagnoses (e.g., pneumonia, lung cancer, or COPD), aligning with the clinician's diagnostic reasoning. This multi-level hierarchy allows radiologists to follow a clear progression from a general abnormality to a very specific diagnosis, making it easier to understand the model's results and assess its reliability. From a scalability perspective, the same hierarchical structure reduces the complexity of large-scale classification tasks by decomposing them into smaller, more manageable subproblems. Instead of learning discriminative features for thousands of flat classes simultaneously, the model progressively narrows down predictions by traversing the hierarchy, improving computational efficiency and making it feasible to handle extensive class taxonomies without degrading performance.

Despite the clear advantages of hierarchical classification, the majority of CNN-based classification methods continue to predominantly rely on flat models, where each class is treated independently, disregarding any relationships between them. In these models, the network must simultaneously differentiate between all possible classes, a process that not only amplifies task complexity but also demands substantial computational resources, especially when handling large number of classes. Furthermore, flat models fail to account for the varying severity of misclassifications within a hierarchical context. A primary reason for the continued preference for flat models is the scarcity of readily available hierarchical datasets. Constructing such datasets is challenging, requiring careful definition of meaningful hierarchies and precise labeling across multiple levels. Additionally, evaluating hierarchical models requires the development of hierarchical performance metrics and loss functions that accurately capture hierarchical class relationships. The lack of structured datasets, appropriate evaluation metrics, and specialized loss functions makes it difficult to train models that exploit class relationships, thus preserving the dominance of flat models in many domains. This paper addresses these limitations by introducing an end-to-end hierarchical classification model built upon the *YOLO* model family. Experimental results highlight how our hierarchical approach overcomes the challenges inherent in flat models, particularly in the real-world task of grocery store item classification.

2 Related Work

Incorporating hierarchical structures into deep learning tasks has proven to be a successful approach in various domains, including computer vision and text classification. Research efforts have largely focused on adapting neural network architectures, loss functions, and label representations to better capture hierarchical relationships between classes.

Architectural adaptations commonly include designs that reflect hierarchical dependencies explicitly: adding multiple output layers corresponding to different levels of the class hierarchy, designing custom layers that encode hierarchical relationships, or integrating graph-based structures. For example, Zhu and Bain (2017) introduced the Branch Convolutional Neural Network (**B-CNN**), which contains multiple branch networks along the main convolutional path, each corresponding to a different level in the class hierarchy. Building on this, Taoufiq, Nagy, and Benedek (2020) replaced parallel branches with a coarse-to-fine classification strategy and introduced a multiplicative layer to explicitly model dependencies between coarse and fine predictions, resulting in fewer parameters and improved hierarchical consistency in their **HierarchyNet**. With **Tree-CNN** Roy, Panda, and Roy (2020) further advanced hierarchical modeling by organizing classifiers in a tree structure that can grow incrementally as new classes appear, enabling scalable learning without retraining from scratch. Another architecture-level innovation by Zunaed and Fattah (2022) proposed classifier-block-level hierarchies, reducing redundancy and memory consumption relative to traditional network-level hierarchical models. An alternative approach by Grassa, Gallo, and Landro (2021) modified the output layers of a standard ResNet18, incorporating multiple linear layers for different hierarchy levels, and combined cross-entropy loss which aims to maximize inter-class variance, with center loss (Wen, Zhang, Li, and Qiao (2016)) to minimize intra-class variance. These architectural innovations demonstrate a range of methodologies for embedding hierarchical awareness directly into network design, improving classification accuracy, scalability, and adaptability to complex label taxonomies. Building on this foundation, our novel architecture integrates multiple hierarchical layers, performing convolutional operations at each level to effectively capture class dependencies and enhance hierarchical consistency throughout the model.

In parallel, **loss function engineering** has addressed the challenge of incorporating hierarchical information to improve model training. Muller and Smith (2020) proposed a Hierarchical Loss for semantic segmentation that penalizes errors according to the semantic distance between predicted and true labels, encouraging semantically plausible mistakes when exact classification cannot be achieved. In a related approach, Bertinetto, Müller, Tertikas, Samangooei, and Lord (2020) designed a Hierarchical Cross-Entropy Loss that increases penalties for misclassifications across distant hierarchy branches. Kobayashi (2021) proposed a hierarchy-aware training approach using soft hierarchical targets to share information among related classes, improving generalization especially for imbalanced datasets. Goyal, Choudhary, and Ghosh (2021) developed a Hierarchical Class-Based Curriculum Loss, guiding the model to learn coarse distinctions before fine-grained ones by using the hierarchy as a curriculum. These methods collectively illustrate diverse strategies—from pixel-level

penalty adjustment (Muller and Smith (2020)) and curriculum learning (Goyal et al. (2021)) to severity-weighted loss (Bertinetto et al. (2020)) and soft target regularization (Kobayashi (2021)) — to integrate hierarchical structures into loss functions and model training.

Focusing specifically on **hierarchical object classification with *YOLO* based models**, Redmon and Farhadi (2017) extended the original *YOLOv2* model to *YOLO9000*, to classify a significantly larger number of classes, up to 9000, by introducing the WordTree hierarchical concept. To classify a specific object, the model calculates the conditional probability at each node level and then traverses the tree from the target node up to the root, multiplying these probabilities along the way to obtain the final class probability. While pioneering, *YOLO9000* has limited flexibility due to its reliance on a fixed, coarse-grained WordTree hierarchy, which may not align with domain-specific taxonomies. Additionally, it lacks hierarchy-aware loss functions and intermediate-level supervision, and is built on the outdated *YOLOv2* architecture, which falls short in accuracy and efficiency compared to modern models. As an application-specific extension, Kalhagen, Olsen, Goodwin, and Gupta (2022) proposed a *YOLO* FISH hierarchical model to identify fish species in underwater video feeds and classify them in seven classes. This model adapts the *YOLO9000* framework by modifying the Non-Maximum Suppression (NMS) technique to remove redundant bounding boxes irrespective of object class. Furthermore, the *YOLOv3* detection layers were substituted with those from *YOLO9000*, enhancing the model's ability to detect hierarchies.

The broader *YOLO* framework has undergone rapid evolution in recent years. From *YOLOv3* through *YOLOv5*, improvements centered on multi-scale prediction, CSPNet-based architectures, and performance optimizations in both accuracy and inference speed. *YOLOv6* and *v7* introduced dynamic label assignment strategies and more efficient training routines. *YOLOv8* brought architectural refinements such as decoupled head structures, anchor-free detection, and increased modularity (Terven, Córdova-Esparza, and Romero-González (2023)). Despite these advancements, hierarchical classification focused on comprehensive class taxonomies remains largely underdeveloped within the *YOLO* family, with existing examples typically confined to narrow, domain-specific applications rather than forming a generalized framework. For example, Usmani, Mahmood, Elmadany, Azeem, and Zualkernan (2025) introduced Hierarchical *YOLO* with Real-Time Text Recognition for UAE traffic signs, combining hierarchical detection with embedded text recognition to improve accuracy in complex environments. Iwano, Shibuya, Kagiwada, and Iyatomi (2024) proposed a Hierarchical Object Detection and Recognition Model for practical plant disease diagnosis, using hierarchical structures to improve robustness and interpretability. Recent advancements have mostly concentrated on incorporating hierarchical context into feature extraction, rather than developing models that perform classification across hierarchical class structures. HCA-YOLO (Feng et al. (2024)) embedded a hierarchical coordinate attention mechanism within *YOLOv8*'s backbone to enhance multi-level feature representation. Similarly, HGO-YOLO (Q. Zheng et al. (2025)) enhances *YOLOv8* by integrating HGNetv2 within the backbone to perform hierarchical feature extraction. Though promising, such methods focus on hierarchical features rather

than hierarchical class labels. Our method addresses this gap by introducing a flexible, domain-adaptable hierarchical classification structure alongside an updated *YOLO* backbone, to improve detection accuracy and adaptability across diverse application areas.

The **evaluation of hierarchical classification models** has also become an important area of research, with growing attention to metrics that account for the structure of class hierarchies. Kiritchenko, Matwin, Nock, and Famili (2006) introduced a measure, which considers distance and depth in the class hierarchy, crediting partially correct classifications and discriminating between different types of errors. In our implementation within *YOLO*, we employed this measure to assess and compare the performance of different architectures of the hierarchy. Another hierarchical measure proposed by Kosmopoulos, Partalas, Gaussier, Paliouras, and Androutsopoulos (2015) evaluated pairs of predicted and true classes, assigning costs based on hierarchical distances. In addition, class relationships were represented as a network flow problem, minimizing classification error by pairing classes optimally.

Recent research in hierarchical computer vision has focused on developing scalable, modular models that effectively capture class dependencies across multiple levels of abstraction through varied core mechanisms. For instance,Wang and Barbu (2023) introduce Hierarchical PPCA, which trains independent Probabilistic PCA models per class and clusters them into super-classes, significantly reducing classification complexity and speeding inference for large-scale datasets. In contrast, Mayouf and de Saint-Cyr (2022) present a unified CNN architecture that simultaneously predicts labels at all hierarchy levels using Bayesian adjustments to encode class dependencies and a semantic loss to enforce hierarchical consistency. Huo et al. (2024) integrate CNN and Transformer branches within a multi-scale hierarchical framework, using an adaptive fusion module to combine local and global features, improving accuracy on medical image classification. Extending the use of attention mechanisms, a triplet attention-based model for robotic perception by Bhayana and Verma (2024) introduces hierarchical supervision by jointly predicting object class and hierarchical position, thus enhancing performance on structured datasets. Despite differences in modeling choices, from probabilistic and CNN-based methods to attention and transformer fusion, all these approaches maintain hierarchical consistency and improve efficiency in multi-level classification tasks.

3 Methods

By implementing a natural hierarchical object organization, the proposed end-to-end hierarchical model for image detection and classification, hYOLO, based on *YOLOv8* (Jocher, Chaurasia, and Qiu (2023)), offers several innovative contributions.

Firstly, the hierarchical architecture of hYOLO captures the inter-class relationships between objects by organizing them into a meaningful hierarchical structure. This structure allows the model to understand the contextual relationships between different classes of objects, enabling more accurate and contextually relevant classifications.

Secondly, the modified loss function is designed to penalize errors based on their severity within the hierarchy. A misclassification of a bottle of Reduced-Fat Milk (2% fat) as Whole Milk (3% fat) is penalized less than mistaking a bottle of Reduced-Fat Milk for wine. Thus, by incorporating this hierarchical penalty scheme into the loss function, the model learns to prioritize more critical distinctions, leading to improved overall performance.

In addition, a performance metric that reflects the hierarchical nature of the classification task is implemented. It provides a more nuanced evaluation of model performance by considering the hierarchical relationships between classes.

The implementation of the proposed model requires minor adjustments to the existing *YOLO* framework, such as adapting the input label format to reflect hierarchical relationships and incorporating hierarchical performance metrics for each hierarchical level in the output file. Apart from these adjustments, the model training and evaluation procedures remain consistent with the original *YOLO* setup. However, due to its novel architecture, hYOLO model must be trained independently; utilizing an already trained *YOLO* model and retraining it is not feasible.

3.1 Hierarchical Architectures

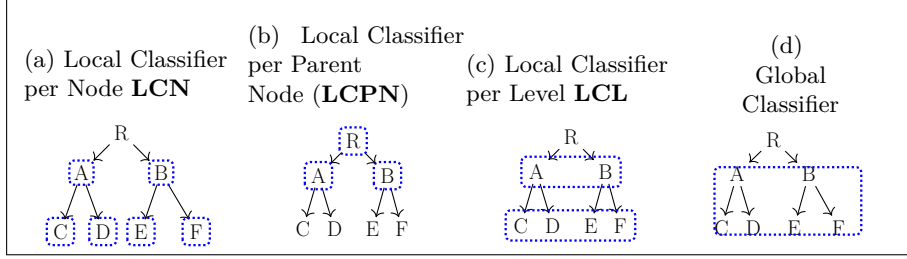A class taxonomy can be referred to as a hierarchy when its structure satisfies the following conditions:

1. It starts with a single root node or top-level class that contains all other classes within the taxonomy. From this root node, the hierarchy branches out into multiple levels, each level representing a subset of classes;
2. Classes are arranged hierarchically, each class has one or more child classes that inherit properties or characteristics from their parent class. This establishes a hierarchical relationship between classes;
3. The structure is acyclic, i.e., two classes cannot be each others ancestor, there are no loops in the hierarchy. This ensures that each class is uniquely positioned within the hierarchy;
4. The structure is anti-reflexive, i.e., a class cannot be a parent of itself.

The most popular hierarchical structures are Directed Acyclic Graphs (DAGs) which allow for multiple paths to the same node, and trees which ensure a unique path from the root to any specified node. For our hierarchy, we opted for a tree-based classifier due to its simplicity and ease of comprehension.

Another important aspect to consider when developing a hierarchical classifier is the way in which the structure is navigated. The most frequently used local classifiers, shown in Figure 1 are:

- Local Classifier per Node (LCN): applies a binary classifier for each node of the hierarchy (Figure 1a),
- Local Classifier per Parent Node (LCPN): assigns a separate multi-class classifier for each parent node (Figure 1b), and
- Local Classifier per Level (LCL): assigns one multi-class classifier for each hierarchical level (Figure 1c).
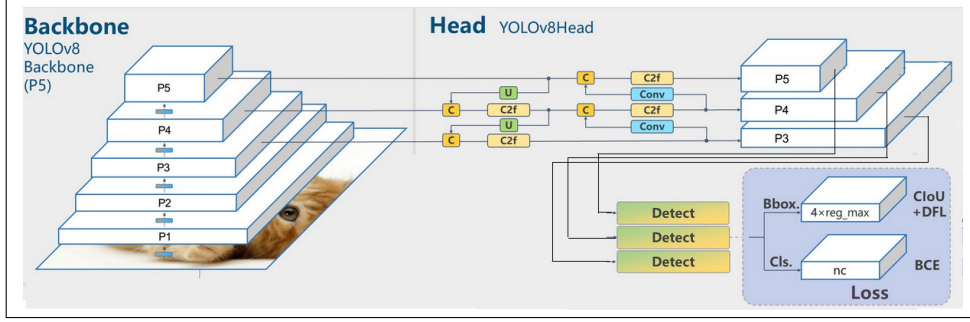
7

Fig. 1: Most popular types of classifiers.

To reduce model complexity and improve generalization LCL was chosen over the other alternatives. However, for the current level the predictions are not restricted solely to the subset of classes predicted at the previous level, thus giving the model the chance to learn and correct potential misclassifications at deeper levels. At each hierarchical level, the number of classes corresponds to the number of categories (nodes). With deeper advancement into the hierarchy, the categories become increasingly refined, leading to a larger number of classes. Ultimately, at the final level, all classes are included analogous to a flat classifier.

To support this hierarchical classification scheme, specific architectural modifications were made to the *YOLOv8* framework. Figure 2a illustrates the head module of the *YOLOv8* architecture (Yaseen (2024)), which is responsible for producing the final predictions: the bounding box coordinates, confidence scores, and class labels. In addition, Figure 2b provides a detailed breakdown of the "Detect" component, explicitly highlighting the exact location within the classification branch where the hierarchical layers were integrated. The bounding box prediction pipeline remains unaltered; only the classification pathway is modified to accommodate the new hierarchical structure.
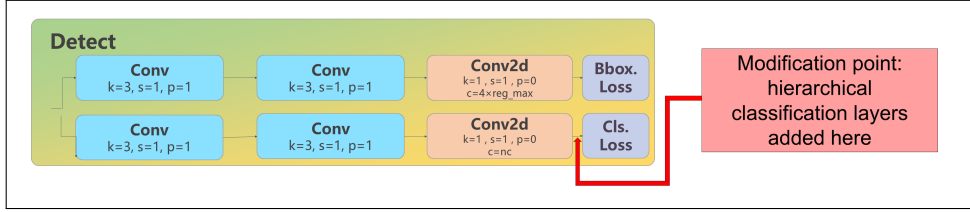
To systematically investigate the optimal strategy for integrating hierarchical information, six alternative hierarchical architectures were explored, each differing in two main aspects: (1) the inserting point, where the hierarchical layers are inserted in the network, and (2) the concatenation point, where outputs from the preceding hierarchical level are merged with the current level. This variation was designed to identify the most effective point within the classification head at which hierarchical information should be integrated — ensuring that the signal passed from the preceding level is both maximally informative and optimally propagated backward during training, thus improving the flow of gradients and feature learning across hierarchical levels. The insertion point, shown in red in Figure 2a applies to versions 1, 2, 4, and 6. In versions 3 and 5, the hierarchical layers are inserted at earlier stages in the classification branch. A full comparison of the architectures is provided in the Experiments Section 4.1.

Here, we focus our analysis on architecture *Version 4* (Figure 3), which demonstrated the highest classification performance across evaluation metrics. To better understand its design and functionality, consider a hierarchical architecture consisting of three distinct levels, where the classification tasks are distributed as follows: *level 0* predicts 2 classes, *level 1* predicts 10 classes, and *level 3* predicts 20 classes.

(a) YOLOv8 head architecture, illustrating the components responsible for bounding box regression and classification.
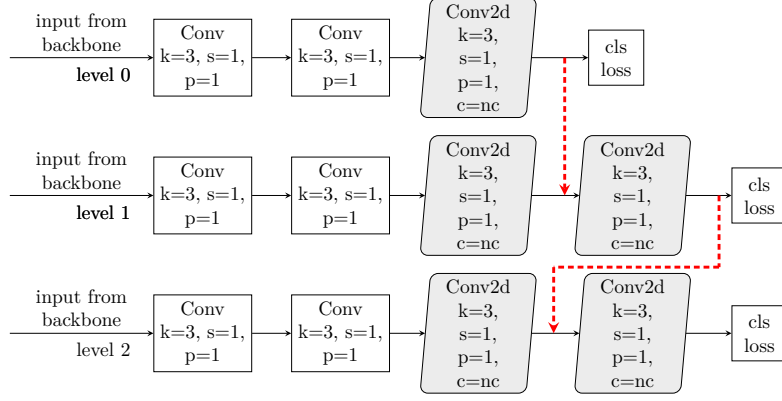

(b) Detection module of *YOLOv8* , annotated with a red arrow to highlight the designated insertion point for the proposed hierarchical classification layers.

Fig. 2: Overview of YOLOv8 head and detection modules, adapted from DL-Diagram repository illustrating both the architecture and proposed insertion points for hierarchical classification layers.

First, the initial input, derived from the backbone of the *YOLOv8* model, is replicated three times, thereby providing an independent input stream for each hierarchical level.

At *level 0*, the Conv2D layer's number of output channels corresponds directly to the 2 target classes. Since only information specific to this current level is available at this stage, classification is performed in a flat manner, consistent with the standard approach employed in *YOLOv8* . For *level 1*, the input is again sourced from the backbone; however, after the first Conv2D operation, feature information from the preceding hierarchical level (*level 0*) is integrated. This fusion allows *level 1* to incorporate the results from *level 0* alongside its own features, thereby embedding contextual information from the preceding classification stage.

Moreover, a second Conv2D layer is introduced at *level 1* (and at each subsequent level) to refine the combined feature representation. This supplementary layer serves to adapt and transform the merged inputs: both the raw backbone features and the propagated information from the previous level, addressing the increased number of classes at this stage. By doing so, the network ensures that the hierarchical information is effectively used and that the subsequent predictions at *level 1* are informed by prior classification results.

9

Fig. 3: Hierarchical architecture $V4$ implemented in *YOLOv8* .

This mechanism of information propagation continues recursively through the hierarchy. Each successive level receives not only the original input from the backbone but also enriched feature representations that encapsulate the predictions from all preceding levels. Since the bounding boxes remain constant across the hierarchical levels, the same physical object is progressively classified with increasing specificity: for example, it may be identified as "food" at *level 0*, refined to "bottle" at *level 1*, and further specified as "bottle of milk" at *level 2*. This hierarchical classification framework thus enables the model to leverage shared spatial information while incrementally enriching the semantic detail of the predictions at each successive level.

3.2 Hierarchical Models Evaluation Metrics

Metrics such as precision, recall, accuracy, or $F1$ score are commonly utilized when evaluating a classification model (Ferrer (2022), Tharwat (2020)). However, these traditional metrics lack the ability to discriminate between different types of misclassification errors in hierarchical classification, since they ignore the relationships between classes (Kiritchenko et al. (2006), Kosmopoulos et al. (2015), Riehl, Neunteufel, and Hemberg (2023)). In hierarchical classification scenarios, it is often more desirable to misclassify an instance into a proximate category rather than a distant one. To address this challenge and effectively assess the model performance of the architectures described in the previous section, we adopted the hierarchical metric proposed by Kiritchenko et al. (2006). A fundamental concept in this metric is the set of ancestors of a given node C, which comprises all nodes lying on paths leading to C, excluding the root node. This metric assesses the distance between the actual class and the prediction, taking into account the number of their common ancestors. It distinguishes between misclassifications in the same subgraph or in a remote subgraph and applies penalties accordingly.

Thus, the conventional evaluation metrics, namely precision, recall, and F_β score, are modified to capture the hierarchical structure of the classification. The modified hierarchical precision (Prec_{Hier}), hierarchical recall (Rec_{Hier}), and hierarchical F_β ($F_{\beta,\text{Hier}}$) are defined with the following equations:

$$\text{Prec}_{Hier} = \frac{\mid \text{Ancest}(C_p) \cap \text{Ancest}(C_t) \mid}{\mid \text{Ancest}(C_p) \mid} \tag{1}$$

$$\text{Rec}_{Hier} = \frac{\mid \text{Ancest}(C_p) \cap \text{Ancest}(C_t) \mid}{\mid \text{Ancestor}(C_t) \mid} \tag{2}$$

$$F_{\beta,\text{Hier}} = \frac{\mid (\beta^2 + 1) * \text{Prec}_{Hier} * \text{Rec}_{Hier} \mid}{\mid (\beta^2 * \text{Prec}_{Hier} + \text{Rec}_{Hier}) \mid} \tag{3}$$

where:

C_p is the predicted class
C_t is the ground-truth class
$\beta \in [0, \infty)$, by default $\beta = 1$
the $\mid \ldots \mid$ denotes the number of elements in the set

Prec_{Hier} and Rec_{Hier} are calculated based on the number of common ancestors between the predicted and ground-truth classes. This value is then divided either by the total number of ancestors of the predicted class or by the total number of ancestors of the ground-truth class. The hierarchical F_β score allows flexible weighting of Prec_{Hier} and Rec_{Hier}. In the most commonly used case, when $\beta = 1$, the measure reduces to the hierarchical $F1$ score ($F1_{Hier}$), which assigns equal importance to precision and recall.

As an illustration, examine the hierarchy in Figure 4, where the ground-truth node $C_t = L$. Consider three potential scenarios for misclassification:

1. If the predicted node $C_p = M$ is in the same subgraph as the ground-truth node, $\text{Ancest}(C_p) = \text{Ancest}(M) = \{B, F, M\}$, $\text{Ancest}(C_p) \cap \text{Ancest}(C_t) = \{B, F\}$, and the number of elements in the intersection $|\text{Ancest}(C_p) \cap \text{Ancest}(C_t)| = |\{B, F\}| = 2$. The total number of ancestors for C_t is $|\{B, F, L\}| = 3$, and for $C_p = M$ is $|\text{Ancest}(M)| = |\{B, F, M\}| = 3$. Then $\text{Prec}_M = \text{Rec}_M = F1_M = 2/3$.

2. If the predicted node $C_p = N$ is in a different subgraph with one common ancestor with the ground truth, $\text{Ancest}(C_p) = \text{Ancest}(N) = \{B, G, N\}$, $\text{Ancest}(C_p) \cap \text{Ancest}(C_t) = \{B\}$, and the number of elements in the intersection $|\text{Ancest}(C_p) \cap \text{Ancest}(C_t)| = |\{B\}| = 1$.) The total number of ancestors for $C_p = N$ is $|\text{Ancest}(N)| = |\{B, G, N\}| = 3$. Then $\text{Prec}_N = \text{Rec}_N = F1_N = 1/3$.

3. If the predicted node $C_p = P$ and the ground-truth node do not share any common ancestors, all Prec_{Hier}, Rec_{Hier} and $F1_{Hier}$ for P are 0.

If a flat classifier is used, all three scenarios result in metrics of 0, regardless of how closely the prediction aligns with the ground truth in terms of hierarchy. In contrast, this modified metric imposes lesser penalties for misclassifications occurring within the same subgraph as the ground truth (mistaking M for L) compared to misclassifications across different and more distant subgraphs (mistaking L for N or P).

The hierarchical architectures outlined in the preceding subsection were evaluated using this novel metric. Among them, $V4$ (Figure 6a) showed the highest $F1_{Hier}$ score.
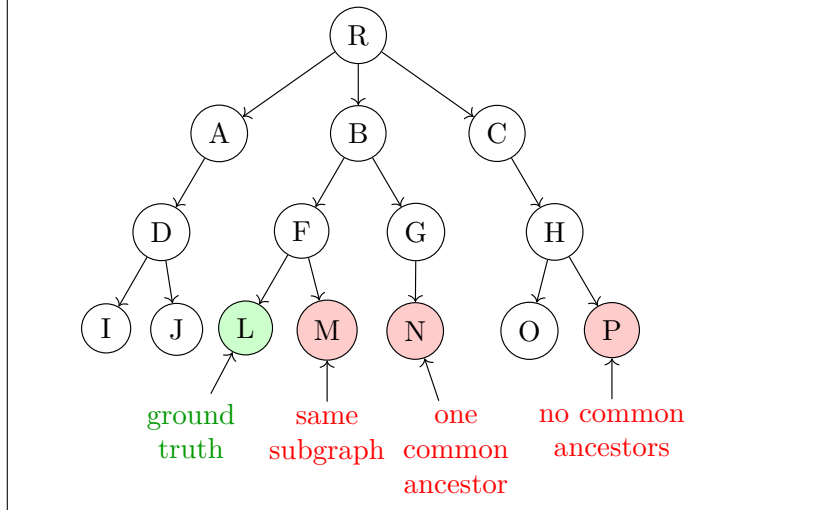
Fig. 4: A tree structured hierarchy.

3.3 Modified Loss Function

The loss function used in *YOLOv8* consists of two key components: a classification term (the standard Binary Cross Entropy Loss), and a bounding box (regression) term. The latter itself is a combination of two independent losses: Distribution Focal Loss (DFL) proposed by Li et al. (2020) and Complete Intersection over Union (CIoU) loss presented by Z. Zheng et al. (2020). Originally designed to address the class imbalance problem in object detection tasks, in *YOLOv8* DFL is also used to improve bounding box regression, especially for difficult to predict objects with blurry or unclear boundaries. On the other hand, CIoU loss considers the aspect ratio differences between the predicted and ground-truth boxes in addition to the overlap between them. The final loss is a weighted sum of these three individual losses (Equation 4a).

$$\mathcal{L} = \mathcal{L}_{\text{Reg}} + \mathcal{L}_{\text{Cls}} \tag{4a}$$

$$\mathcal{L}_{\text{Reg}} = w_{\text{box}} \cdot \mathcal{L}_{\text{CIoU}} + w_{\text{dfl}} \cdot \mathcal{L}_{\text{DFL}} \tag{4b}$$

$$\mathcal{L}_{\text{Cls}} = w_{\text{cls}} \cdot \mathcal{L}_{\text{BCE}} \tag{4c}$$

where:

w_{box} − weight of CIoU loss with default value of 7.5 in *YOLOv8*
w_{dfl} − weight of DFL loss with default value of 1.5 in *YOLOv8*
w_{cls} − weight of BCE loss with default value of 0.5 in *YOLOv8*

To train the neural network to follow the hierarchical structure during object classification, modifications were made to *YOLO*'s loss function. First, for each hierarchical level a distinct loss function is calculated. It is the weighted sum of the individual CIoU, DFL and BCE losses for this particular level. The weights w_{box}, w_{dfl}, and w_{cls}

are constants and set to their default values at each level to maintain a consistent contribution from each loss component to the total loss. Finally, the total loss for the entire hierarchical structure is defined as the average of these level-specific losses. This ensures that the model is trained comprehensively across all hierarchical levels.

$$\mathcal{L} = \sum_{i=0}^{l}(\mathcal{L}_{Regression_i} + \mathcal{L}_{Class_i}) \tag{5}$$

where:

l – hierarchy depth, number of hierarchical levels.

Furthermore, an additional term to penalize predictions which are not children of the parent at the previous hierarchical level was incorporated in the classification loss. As a result, the modified loss function is hierarchy-aware, penalizing both classification errors and violations of the hierarchy.

For the first hierarchical level no penalty term is applied, since there are no parent classes for this level.

$$\mathcal{L}_{Cls_0} = w_{\text{cls}} \cdot \mathcal{L}_{BCE_0} \tag{6}$$

For each subsequent hierarchical level if the predicted class is a child of the parent class on the previous hierarchical level, no penalty is applied. However, if the predicted class is not a child of the parent node, the penalty term is set to be the confidence score of the predicted class at the current level. Consequently, the modified classification loss is computed as follows:

$$\mathcal{L}_{Cls_l} = w_{\text{cls}} \cdot \left(\mathcal{L}_{BCE_l} + \alpha \cdot \sum_{i=1}^{S}(1 - \delta_{il}) \cdot \text{conf}_{il}\right) \tag{7}$$

where:

l – hierarchy depth, number of hierarchical levels
S – number of classes
$\mathcal{L}_{BCE_l}$ – Binary Cross Entropy at hierarchical *level l*
α – regularization constant $\alpha \geq 0$
$\delta_{il} = \begin{cases} 1, & \text{if class } i \text{ is a child of its parent at level } l-1 \\ 0, & \text{otherwise} \end{cases}$

Penalizing an incorrect prediction with its confidence score has an additional advantage: the higher the confidence of the incorrect prediction, the harsher the penalty. Conversely, for incorrect predictions with lower confidence the penalty is less severe. If the regularization constant is set to $\alpha = 0$, the penalty term is completely omitted. In this case, the loss function reduces to the standard BCE loss, and no hierarchical consistency enforcement is applied at that level.
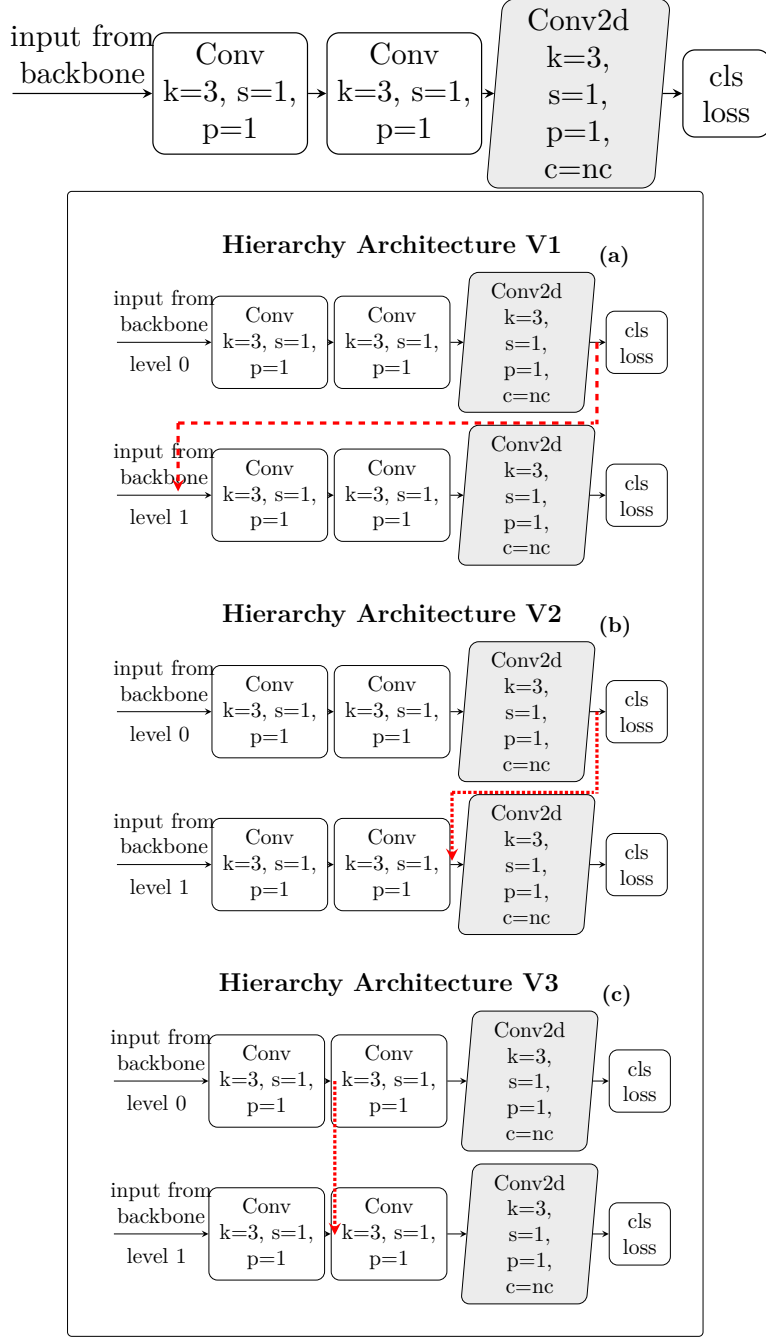
4 Experiments

4.1 Experimental Variants of Hierarchical Architectures

To identify the most effective strategy for integrating hierarchical information, six alternative network architectures were explored. Each design varied in two aspects: the insertion point of the hierarchical branches and the specific location where outputs from preceding levels are concatenated with the current level's features. In all configurations, the output generated by the *YOLOv8* backbone is initially replicated to match the number of hierarchical levels defined in the model. Then, at each level, the output from the previous hierarchical level is concatenated with the current level at a specific point. The dashed red arrows in Figure 5 and Figure 6 indicate these hierarchical connections between levels. The starting point of each arrow represents the layer where the additional hierarchical branch is introduced, while the endpoint signifies the location at the current level where information from the preceding level is concatenated with the current layer's features. The different architectural designs were guided by the principle that performance gains in hierarchical models depend on the point at which different levels of abstraction are merged. If lower-level features (e.g., spatial or edge-based) are combined too late in the network, the model may lose valuable spatial context. Conversely, fusing these features too early can overload the classifier with low-level noise, thereby obstructing the learning of high-level semantics (Zhang, Zhang, Peng, Xue, and Sun (2018)).

In *Version 1* (Figure 5 a), the hierarchical structure is introduced after the Conv2d layer at *level 0*, following a series of convolutional operations through which the model has already extracted low-level features and begun to learn meaningful representations of the input data. However, the concatenation with the subsequent hierarchical *level 1* occurs too early (directly after the backbone output), potentially disrupting the development of higher-order feature representations by introducing prematurely merged signals. This premature fusion may contribute to the notably lower average $F1_{Hier}$ observed for this variant, as reported in Table 1, suggesting that the timing of integration plays a critical role in effective hierarchical learning.

In the evaluation analysis presented in Table 1, we compare the six hierarchical model variants (*Version 1* through *6*) using the hierarchical $F1$ score ($F1_{Hier}$) at each level of the hierarchy. A consistent decline in $F1_{Hier}$ is observed from *level 0* to *level 3* across all models, reflecting the increasing complexity and difficulty of fine-grained classification. To identify the most effective architecture, we use two criteria: the $F1_{Hier}$ score at *level 3* which includes the complete set of classes and represents the most granular level of classification, and the $F1_{Hier}$ score of the worst-performing class at this level to capture edge-case performance. This dual-criteria approach enables a more robust assessment of each architectural variant's capacity to handle the most difficult aspects of the classification task.

Original *YOLOv8* Architecture

Fig. 5: Hierarchical architectures $V1$, $V2$ and $V3$ implemented in *YOLOv8* .

Table 1: Performance evaluation of various architectural versions based on $F1_{Hier}$ metric for the 100-class $V1$ dataset.

Arch. Version	Hier Level	Prec_{Hier}	Rec_{Hier}	Avg. F1_{Hier}	F1_{Hier} Worst Class
1	0	0.9905	0.9954	0.9929	
	1	0.9828	0.9832	0.9830	
	2	0.9788	0.9742	0.9765	
	3	0.9677	0.9584	**0.9630**	0.779
2	0	0.9951	0.9965	0.9958	
	1	0.9839	0.9822	0.9831	
	2	0.9800	0.9731	0.9765	
	3	0.9760	0.9543	**0.9650**	0.738
3	0	0.9947	0.9970	0.9958	
	1	0.9878	0.9873	0.9876	
	2	0.9828	0.9782	0.9805	
	3	0.9730	0.9670	**0.9700**	0.800
4	0	0.9961	0.9971	0.9966	
	1	0.9890	0.9876	0.9883	
	2	0.9833	0.9774	0.9804	
	3	0.9734	0.9686	**0.9710**	**0.814**
5	0	0.9957	0.9966	0.9962	
	1	0.9855	0.9843	0.9849	
	2	0.9821	0.9764	0.9793	
	3	0.9764	0.9598	**0.9680**	0.744
6	0	0.9892	0.9673	0.9781	
	1	0.9887	0.9894	0.9891	
	2	0.9838	0.9822	0.9830	
	3	0.9741	0.9620	**0.9680**	0.701

Building on the insights gained from *Version 1*, modifications aimed at improving the timing of feature integration were explored. In *Version 2* (Figure 5 b), the same insertion point for the hierarchical branch—after the Conv2d layer at *level 0* was kept, but the outputs were concatenated at a later stage: before the Conv2d layer at *level 1*. This adjustment allows the model to learn a richer set of representations at *level 1* prior to the merge. Despite achieving a slightly higher overall $F1_{Hier}$ of 0.965, this version exhibited a lower worst-class $F1_{Hier}$ of 0.738. This suggests that while the delayed concatenation in *Version 2* may slightly enhance global classification performance, it might also suppress critical features for difficult classes, possibly due to delayed access to complementary low-level features.

While *Version 1* suffered from overly premature concatenation and *Version 2* attempted to fix this by delaying fusion, both designs retained the same initial insertion point, potentially limiting their capacity to optimally exploit low-level spatial features. To further investigate the impact of feature timing and semantic alignment, our next architectural variant *Version 3* (Figure 5 c) was designed by shifting both the

insertion point and the concatenation point to earlier stages within the network. The idea behind this modification was to allow each hierarchical level to begin processing contextualized features sooner and in closer proximity to the point of integration. By injecting and fusing information earlier in the feature extraction process, the model is given the opportunity to jointly learn spatial and semantic representations in a coordinated fashion. As shown in Table 1, *Version 3* demonstrated an improvement in both average $F1_{Hier}$ and worst-class performance relative to prior configurations. *Version 5* (Figure 6 a) retained the same insertion and concatenation scheme employed in *Version 3*, but both points were shifted one convolutional layer deeper into the network. Despite this adjustment, the results showed a decline in both the overall $F1_{Hier}$ and the worst-class $F1_{Hier}$, suggesting that the later integration may have blocked the model's ability to maintain discriminative feature representations at finer levels of classification.

In *Versions 4* and *6*, a second Conv2D layer was introduced at *level 1* and at each subsequent hierarchical level. This extra layer serves as a refinement block, ensuring that hierarchical signals are not merely passed along but are processed to resolve conflicting or misaligned representations, and filter out noise or redundancy that may arise from premature or improperly aligned fusion.
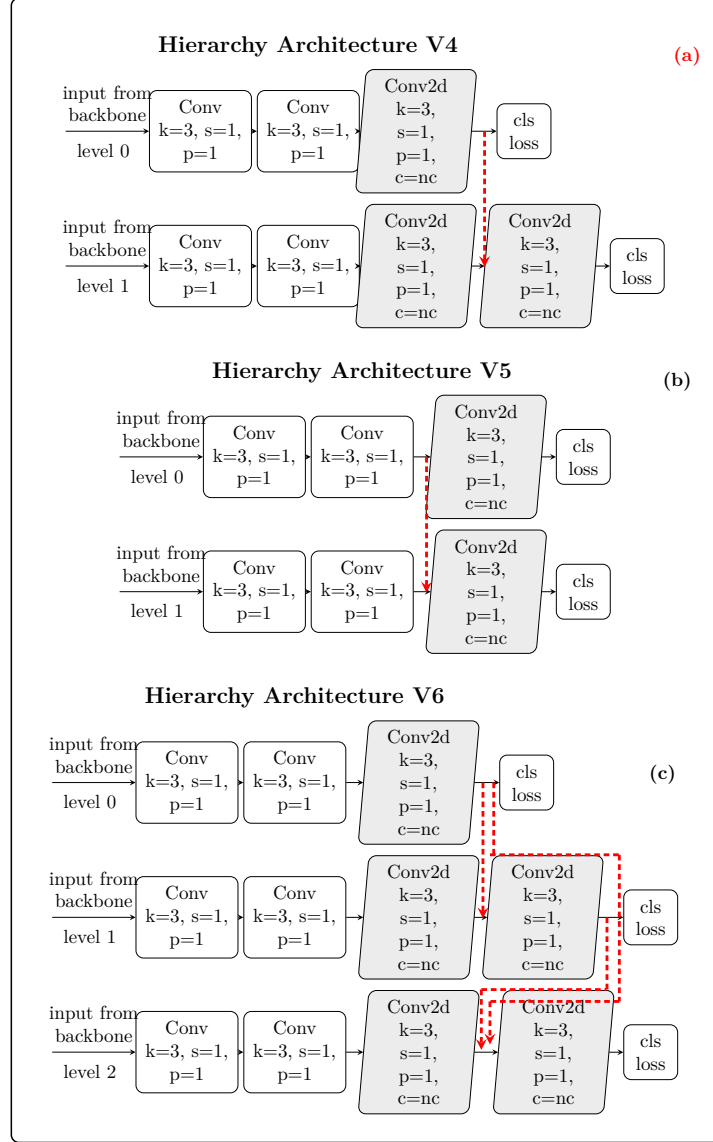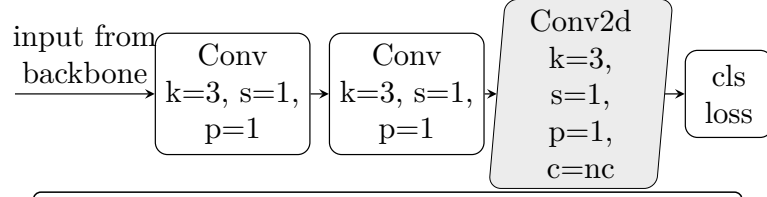
Furthermore, in *Version 6*, multiple concatenation points were introduced, specifically from *level 0* to both *levels 1* and *2*. This design choice aimed to enrich the feature representations at higher levels by directly incorporating lower-level information. By allowing these skip connections, the network can more effectively integrate multi-scale features, potentially improving robustness against information loss or degradation that may occur during hierarchical processing. However, this approach did not yield the desired improvements. While the overall $F1_{Hier}$ score remained unchanged, the $F1_{Hier}$ score for the worst-performing class actually decreased.

Further analysis of Precision and Recall across all six hierarchical architectures (Table 1) revealed only minor variations at each hierarchical level, indicating that all models generally maintained a well-balanced performance. In addition, all models achieved consistently high metrics, particularly at the base hierarchical level, where both Precision and Recall often exceeded 0.99. However, *Version 6* exhibited a reduction in Recall at *level 0*, indicating a higher rate of missed positive instances despite maintaining strong Precision. These findings support the selection of *Version 4* as the optimal model, as it provides a more balanced and robust classification performance across all hierarchical levels and achieves the highest overall and worst-performing class $F1_{Hier}$ scores.

4.2 Datasets

The complete dataset consisted of 78852 images of 1011 grocery store items (classes) and was partitioned into three subsets: a training set of 42456 images, a validation set of 18198 images, and a testing set of 18198 images. The image augmentation process was conducted as follows: for each exposition of a given grocery store item, 17 random expositions of other items were selected. These items were added to the image sequentially, starting from the larger items and proceeding to the smaller ones. Each item was included only if the maximum allowed occlusion of 70% was not exceeded.

Original *YOLOv8* Architecture

Fig. 6: Hierarchical architectures $V4$, $V5$ and $V6$ implemented in *YOLOv8* .

Consequently, each image contained approximately 18 objects on average (Figure B2 in the Appendix).

In addition, the dataset is balanced: each class is represented by approximately 300 instances in both the validation and testing subsets, and by approximately 700 instances in the training subset. The instances were generated through permutations of the 10 different expositions captured by a camera for each item (Figure B3 in the Appendix).

Furthermore, to conserve time and computational resources, a reduced dataset comprising only 100 classes was utilized. This subset was specifically sampled to be statistically representative of the complete dataset and was used in certain experiments to expedite the training process.

4.3 Dataset Hierarchical Structures

Two distinct hierarchical structures of the dataset were developed to evaluate the impact of hierarchical depth and the significance of visual similarities between objects on model performance.
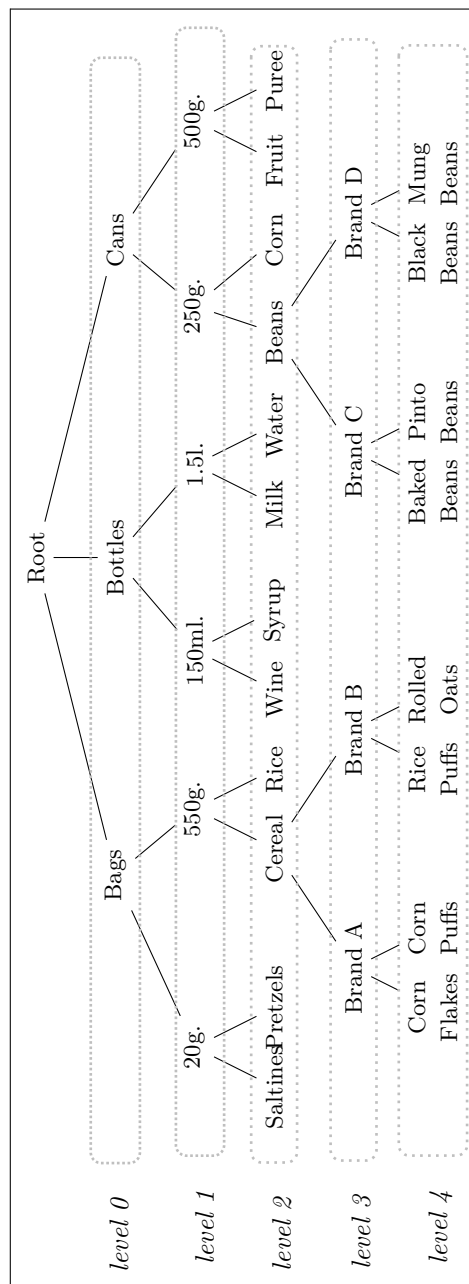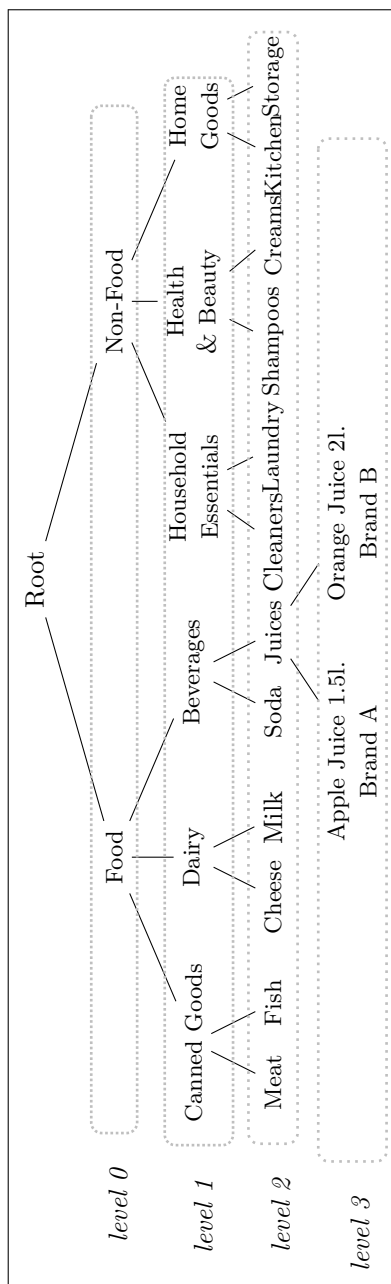
4.3.1 Dataset Hierarchy Version 1:

In this hierarchical structure the class relations were solely based on store categories, disregarding visual resemblances between objects. This decision was made for the sake of convenience, ensuring that the hierarchy followed a typical supermarket categorization system, starting with broad categories and gradually refining towards more detailed classifications of items within the inventory. The classes were distributed across four hierarchical levels as shown in Figure 7. For instance, at *level 0*, classes were divided into food and non-food categories. At *level 1*, the food category was further divided into canned food, dairy, beverages, etc., while the non-food category was divided into household, beauty, home products, etc. Subsequently, at *level 2*, dairy was further subdivided into milk, cheese, yogurt and other specific categories. At the last level, the classes corresponded precisely to the store inventory system nomenclature. This methodology required no additional mapping between hierarchical model categories and inventory categories, as it directly followed the pre-existing inventory framework.

This hierarchical concept was tested only on the reduced 100-class dataset, since it did not show promise and underperformed compared to the flat *YOLOv8* model. Despite the structured approach, the hierarchical model failed to achieve the desired accuracy, prompting further investigation into alternative hierarchical strategies. This led to the exploration of a hierarchy based on visual similarities between objects.

4.3.2 Dataset Hierarchy Version 2:

In dataset hierarchy version 2 the hierarchical depth was increased to five levels. In addition, visual similarities between items were considered, particularly at lower hierarchical levels. The shape of an object was deemed to be more informative than its size for this classification task due to significant variations in perceived size caused by changes in camera angle, distance, and zoom. Conversely, the shape typically remained

Fig. 7: Dataset Version 1: Categorization with four hierarchical levels, (only a subset of categories shown).

Fig. 8: Dataset Version 2: Categorization with five hierarchical levels.
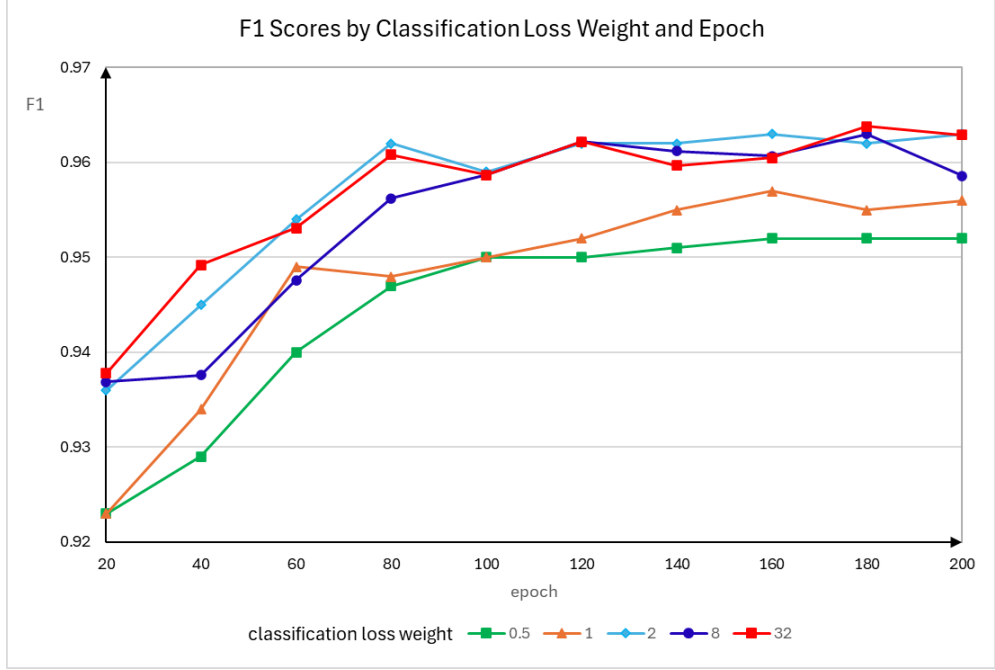
consistent, making it a more reliable feature for classification purposes. Therefore, shape was chosen to be the distinctive feature at the lowest *level 0*, followed by size, type, and brand. For instance, categories at *level 0* which represented the shape of the object, included bag, bottle, box, can, cylinder, etc., while *level 1* indicating the size of the object comprised of categories like $1000g.$, $1000ml.$, $100g.$, $10g.$, etc. Higher levels reflected the store's nomenclature: for example, *level 3* described the type of the product such as toothbrush, water, flour, while *level 4* indicated the manufacturer's name, or brand. Once again, classes at *level 4* precisely matched the store's inventory system nomenclature (Figure 8).

4.4 *YOLO* Model and Hyperparameters

The $V4$ architecture explained in Sec. 3.1, and the modified loss function defined with Equation 7 were both implemented into *YOLOv8*. Pre-trained weights from the COCO dataset were utilized not only to achieve faster convergence and improve model generalization, but also to ensure comparability between models with different architectures. Thus, a standardized starting point for all models was provided, enabling fair comparison across distinct architectures.

While the model hyperparameters remained at their default settings, the classification loss weight parameter w_{cls} was modified to better align the training objective with our task. This modification reflects the understanding that different applications may require varying emphasis on classification versus localization. In the *YOLO* framework: the total loss comprises three components: the Complete IoU (CIoU) loss, which serves a bounding box regression loss, the Distribution Focal Loss (DFL), which refines the localization precision of predicted boxes, and the Binary Cross-Entropy (BCE) loss, which is used for classification. Each component contributes to the final loss according to its designated weight: 7.5 for CIoU (bounding box), 1.5 for DFL, and 0.5 for BCE (classification). These weights reflect the relative importance assigned to localization, objectness confidence, and class prediction during training. The choice of these weights, however, should be task-dependent. In applications focused primarily on object detection, such as autonomous driving, which requires accurate identification of potholes on roadways, the relative importance shifts from classification to localization. In these scenarios, misclassifying a roadside artifact as a pothole is preferable to missing the detection altogether, as the vehicle can still navigate around it. Therefore, increasing the weight of the box loss w_{box} is justified in detection-focused applications, where localization accuracy should be prioritized over classification loss. Ultimately, the tuning of individual loss weights, including w_{cls} and the box loss weights w_{box}, should be guided by the specific objectives of the application.

Given that our application focuses on the fine-grained classification of grocery store items, accurate categorization is more important than precise localization. For instance, even if an object is slightly misaligned, shifted to the left or right, it should still be correctly classified. However, misclassifications such as confusing two visually similar but distinct products, can significantly reduce the system's practical value. To better reflect this priority in the model's training objective, we experimented with increasing the classification loss weight w_{cls} to assess its impact on the $F1$ score. As shown in Figure 9, when training a classification model with the reduced dataset

Fig. 9: Comparison of $F1$ scores on the test dataset across different classification loss weights (100-class dataset).

of 100 classes, the $F1$ score increased with w_{cls}, starting at 0.948 at $w_{\mathrm{cls}} = 0.5$, ultimately reaching its peak of 0.962 at $w_{\mathrm{cls}} = 2$. Therefore, the default value of $w_{\mathrm{cls}} = 0.5$ was changed to 2. In addition, SGD optimizer was favored over Adam since it demonstrated superior stability during training for our particular model and dataset (refer to Section C in the Appendix for a comprehensive overview of the configuration parameters).

4.5 Model Training and Evaluation

Hierarchy Architecture $V4$ was chosen over the alternatives due to its highest scores across all classes as well as in the worst performing class.

Subsequently, the model was trained on the two hierarchical versions of the dataset, described in Sec. 4.3, using the same $V4$ architecture for both. To assess the model's performance, three key metrics were considered: the $F1$ score, True Positive (TP) confidence score, and False Positive (FP) confidence score. The TP confidence score for a specific class is defined by averaging the confidence scores across all true positive predictions associated with that class. Similarly, the FP confidence score is calculated by averaging the confidence scores of all false positive predictions for that specific class. Finally, the TP and FP confidence scores for the entire model are computed as the averages of the TP, or FP scores, respectively, across all classes.

Dataset Hierarchy Version 2, which accounted for the visual and semantic similarities between objects, demonstrated superior performance when applied to the 100-class dataset. This version outperformed both its predecessor, Version 1, and the flat *YOLOv8* model across all three performance metrics: precision, recall, and F1 score (Table 2). In particular, it achieved the highest TP confidence scores, indicating a greater certainty in its correct predictions. Additionally, it recorded the lowest FP confidence score, suggesting that its incorrect predictions were made with relatively lower confidence, thus enhancing its reliability.

To ensure a fair comparison with the flat *YOLOv8* model, which does not naturally support hierarchical classification, five separate flat models were trained, each corresponding to a distinct hierarchical level within the Dataset Version 2 hierarchy. Moreover, because the hierarchical version of the $F1$ metric is not applicable to flat models, standard precision, recall, and $F1$ scores were used for all evaluations to maintain consistency across comparisons.

Table 2: Comparison between V1, V2 dataset Hierarchies of hYOLO and Flat *YOLOv8* Model using the 100-class dataset on $V4$ Architecture. Note that the Flat *YOLOv8* model is evaluated using the V2 dataset.

Model Type	Hier. Level	Precision	Recall	F1 Score	TP Conf	FP Conf
Dataset Version 1	0	0.9961	0.9961	0.9961	0.9555	0.1295
	1	0.9882	0.9830	0.9856	0.9304	0.0978
	2	0.9783	0.9721	0.9751	0.9326	0.0705
	3	0.9552	0.9564	0.9558	0.9179	0.0879
Dataset Version 2	0	0.9922	0.9941	0.9932	0.9592	0.0833
	1	0.9778	0.9784	0.9781	0.9427	0.0855
	2	0.9790	0.9771	0.9781	0.9457	0.0679
	3	0.9748	0.9777	0.9763	0.9448	0.0683
	4	0.9660	0.9648	*0.9654*	*0.9406*	*0.0765*
Flat *YOLOv8*	0	0.9899	0.9919	0.9909	0.9488	0.1548
	1	0.9711	0.9745	0.9728	0.9354	0.1259
	2	0.9835	0.9768	0.9801	0.9374	0.1002
	3	0.9741	0.9722	0.9731	0.9332	0.4123
	4	0.9659	0.9610	0.9635	0.9298	0.1253

When extended to the more complex 1011-class dataset, Dataset Hierarchy Version 2 again exhibited lower FP confidence values and higher TP confidence than the flat model (Table 3). We also analyzed the fraction of false positive predictions that fell within the same subgraph as the ground-truth label. This metric reflects semantic closeness even in the presence of prediction errors. Dataset Version 2 significantly

outperformed the flat model on this dimension achieving a subgraph-level FP fraction of 0.6844 at the last hierarchy level 4, compared to 0.6508 for the flat model.

Interestingly, despite the hierarchical model's superior confidence calibration and semantic consistency, the flat *YOLOv8* model exhibited a higher $F1$ score. However, this comes with the trade-off of increased FP confidence, and a higher likelihood of semantically irrelevant misclassifications, as indicated by the lower fraction of FPs falling within the correct subgraph, underscoring the value of embedding visual similarity into the dataset hierarchy.

Table 3: Comparison between V2 dataset hierarchy of hYOLO and Flat *YOLOv8* Model using the 1011-class dataset on $V4$ Architecture.

Model	HLevel	Prec.	Recall	F1 Score	TP Conf	FP Conf	FP Same Subgraph
Dataset V2	0	0.9914	0.9880	0.9897	0.9423	0.0662	
	1	0.9512	0.9455	0.9484	0.9147	0.0708	
	2	0.9556	0.9530	0.9543	0.9235	0.0449	
	3	0.9660	0.9535	0.9597	0.9299	0.0429	
	4	0.8891	0.8876	0.8883	*0.8735*	*0.0730*	*0.6844*
Flat *YOLOv8*	0	0.9899	0.9919	0.9909	0.9488	0.1548	
	1	0.9711	0.9745	0.9728	0.9354	0.1259	
	2	0.9835	0.9768	0.9801	0.9374	0.1002	
	3	0.9741	0.9722	0.9731	0.9332	0.1089	
	4	0.8976	0.8887	*0.8931*	0.8710	0.0970	0.6508

In subsequent experiments, the loss function was modified by adding a penalty term as defined in Equation 7. Various values of α were tested to evaluate their impact on performance metrics (Table 4). While an α value of 50 may initially appear excessively high, it is, in fact, appropriate since, in some cases, the introduced penalty term can be smaller by an order of magnitude or more compared to the Binary Cross Entropy (BCE) loss. Without appropriately scaling its weight, the penalty's influence on the overall loss function would be negligible.

As shown in Table 4, different values of α optimized different performance metrics. For instance, the best TP confidence score was achieved with $\alpha = 0.9$, whereas the highest $F1$ score and best FP confidence was attained when no penalty term was added. Notably, an increase of α led to an improvement of the $F1$ score; however, the TP score decreased. The lowest per-class $F1$ score increased from 0.362 ($\alpha = 0.5$) to 0.589 for $\alpha = 25$.

Table 4 may initially suggest that the "no penalty" setting ($\alpha = 0$) outperforms penalized variants due to its highest average $F1$ score. However, it also yields a markedly lower $F1$ score on the worst-performing class (0.485), indicating uneven class-wise performance. In contrast, moderate penalty values (e.g., $\alpha = 25$) result in

Table 4: Performance evaluation of hYOLO for different α values on the 100-class dataset.

Alpha	Level	TP conf	FP conf	Avg. F1 Score	F1 Worst Class
0	0	0.9590	0.0830	0.9930	0.9810
	1	0.9430	0.0850	0.9774	0.8810
	2	0.9460	0.0680	0.9771	0.8810
	3	0.9478	0.0680	0.9751	0.8770
	4	0.9410	*0.0760*	*0.9610*	0.4850
0.5	0	0.9594	0.0919	0.9918	0.9790
	1	0.9407	0.1191	0.9729	0.8170
	2	0.9462	0.0982	0.9724	0.8240
	3	0.9478	0.0903	0.9719	0.8210
	4	0.9406	0.1025	0.9543	0.3620
0.9	0	0.9667	0.0893	0.9919	0.9570
	1	0.9530	0.1394	0.9704	0.8100
	2	0.9556	0.1028	0.9716	0.8470
	3	0.9550	0.1366	0.9713	0.8710
	4	*0.9447*	0.1360	0.9528	0.4170
25	0	0.9492	0.0714	0.9899	0.9750
	1	0.9314	0.1711	0.9731	0.8950
	2	0.9382	0.1404	0.9743	0.8800
	3	0.9396	0.1675	0.9724	0.8480
	4	0.9312	0.1566	0.9557	*0.5890*
50	0	0.9461	0.0539	0.9913	0.9830
	1	0.9212	0.1879	0.9762	0.8980
	2	0.9273	0.1481	0.9749	0.8500
	3	0.9350	0.1891	0.9740	0.8430
	4	0.9251	0.1681	0.9566	0.4950

slightly lower average $F1$ scores but offer significantly improved robustness, particularly for difficult classes—evidenced by the highest worst-class $F1$ score (0.589). These findings suggest that the magnitude of the penalty term needs tuning, as different values of α yield varying trade-offs across key performance metrics. Therefore, selecting an optimal α should be guided by the specific requirements of the task, such as the importance of overall average performance versus robustness to rare classes.

5 Conclusion

This paper introduces hYOLO, a hierarchical end-to-end model for image classification tasks, built upon *YOLOv8* . Our approach incorporates a novel hierarchical architecture, modified loss function, and hierarchical labels. We trained and evaluated the model on two hierarchies derived from the same dataset: one based on formal object categorization without considering visual similarities, and another that accounted for visual resemblances between classes. This dual approach allowed us to assess the impact of hierarchical organization on model performance. Our results demonstrate that incorporating a hierarchical structure aware of the visual similarities between classes significantly improves model performance in image detection and classification tasks. This methodology presents itself as a robust framework for future research and application in various domains.

The practical potential of the proposed hYOLO model extends across a wide spectrum of real-world computer vision applications where visual classes follow an inherent hierarchical structure. In **medical imaging**, hierarchical classification enables more nuanced differentiation between visually similar disease subtypes, such as various forms of tumors, supporting more accurate diagnostics and assisting decision-support systems in identifying rare or emerging conditions within broader pathological families. In **retail environments**, applications such as smart shopping carts, self-checkout systems, and inventory management platforms benefit from hierarchical product classification (e.g., *fruit → citrus → orange*) (Wei, Tran, Xu, Kang, and Springer (2020)). This structure improves product recognition under occlusion or varying packaging, while also enhancing robustness to changes in viewpoint, lighting conditions, and partial visibility during real-time use. In **autonomous driving**, understanding hierarchies such as *vehicle → truck → delivery truck* or *sign → warning sign → pedestrian crossing* supports more reliable object recognition and context-aware decision-making, contributing to safer navigation and environment-aware behavioral planning. In **intelligent surveillance and public safety**, hierarchical modeling can improve both precision and interpretability. By recognizing visual similarities within broader semantic groupings—e.g., *tool → hand tool → hammer* versus *weapon → firearm → handgun*, the system can reduce false positives, differentiate harmless from threatening objects, and enhance situational awareness in real time. This is particularly valuable in high-stakes environments such as airports, public venues, or critical infrastructure surveillance, where accurate threat detection is important. In **e-commerce image classification**, hierarchical understanding may allow platforms to better manage extensive product taxonomies. For instance, classifying a product as *clothing → outerwear → jacket → leather jacket* enables more relevant search results, improved recommendations, and more granular filtering options (Seo and Shin (2019)). This will improve user experience and will facilitate more efficient product discovery across vast and diverse online catalogs. In **ecological and environmental monitoring**, the classification of flora, fauna, and land cover types aligns naturally with taxonomic hierarchies (e.g., *animal → bird → raptor → eagle*), thus supporting scalable biodiversity assessment, species monitoring, and habitat mapping, which is essential for conservation science (Ojwang et al. (2023)). Finally, in **digital art and cultural heritage preservation**, classifying artifacts by attributes such as *art → painting →*

Renaissance → Italian may enhance digital curation, retrieval, and archival efforts in museums and academic institutions (Jain et al. (2020) and Neudecker (2022)). This will facilitate more structured metadata creation and will support automated tagging for improved user interaction and historical analysis.

The promising results achieved with our hierarchical model highlight several avenues for future research, including exploring deeper hierarchies, alternative architectures, enhanced loss functions, establishing standardized guidelines for creating hierarchical datasets and best practices for defining hierarchical levels, categories, and labeling conventions, developing tools and metrics for evaluating the quality of hierarchical datasets, and addressing the challenge of class imbalance within hierarchical datasets, particularly at deeper levels of the hierarchy. By following these research paths, further advancement of hierarchical modeling will be achieved, ensuring robust and reliable performance in practical applications.

6 Data Availability

The datasets generated and analyzed in this study are not publicly available due to proprietary restrictions. The data constitute a private dataset, created with significant investment in its generation, and are subject to copyright and intellectual property protections. Sharing the complete dataset would violate proprietary considerations. However, we provide a representative subset containing 100 classes, which reflects the diversity and structure of the full dataset and is sufficient to reproduce the results presented in this paper. This datsaset is available upon reasonable request.

7 Code Availability

The code to reproduce the experiments described in this paper is available at the following repository: https://github.com/ds2run/hyolo

References

Bertinetto, L., Müller, R., Tertikas, K., Samangooei, S., Lord, N.A. (2020). Making better mistakes: Leveraging class hierarchies with deep networks. *2020 IEEE/CVF conference on computer vision and pattern recognition, CVPR 2020, seattle, wa, usa, june 13-19, 2020* (pp. 12503–12512). Computer Vision Foundation / IEEE. Retrieved from https://openaccess.thecvf.com/content_CVPR_2020/html/Bertinetto_Making_Better_Mistakes_Leveraging _Class_Hierarchies_With_Deep_Networks_CVPR_2020_paper.html

Bhayana, D.A., & Verma, O.P. (2024). Triplet attention-based deep learning model for hierarchical image classification of household items for robotic applications. *Signal Image Video Process.*, *18*(S1), S489–S498, https://doi.org/10.1007/S11760 -024-03168-3 Retrieved from https://doi.org/10.1007/s11760-024-03168-3

Chen, H., Miao, S., Xu, D., Hager, G.D., Harrison, A.P. (2020). Deep hiearchical multi-label classification applied to chest x-ray abnormality taxonomies. *Medical Image Anal.*, *66*, 101811, https://doi.org/10.1016/J.MEDIA.2020.101811 Retrieved from https://doi.org/10.1016/j.media.2020.101811

Feng, Y., Huang, J., Du, S., Ying, S., Yong, J., Li, Y., . . . Gao, Y. (2024). Hyperyolo: When visual object detection meets hypergraph computation. *CoRR*, *abs/2408.04804*, , https://doi.org/10.48550/ARXIV.2408.04804 Retrieved from https://doi.org/10.48550/arXiv.2408.04804 2408.04804

Ferrer, L. (2022). Analysis and comparison of classification metrics. *CoRR*, *abs/2209.05355*, , https://doi.org/10.48550/ARXIV.2209.05355 Retrieved from https://doi.org/10.48550/arXiv.2209.05355 2209.05355

Goyal, P., Choudhary, D., Ghosh, S. (2021). Hierarchical class-based curriculum loss. Z. Zhou (Ed.), *Proceedings of the thirtieth international joint conference on artificial intelligence, IJCAI 2021, virtual event / montreal, canada, 19-27 august 2021* (pp. 2448–2454). ijcai.org. Retrieved from https://doi.org/10.24963/ijcai.2021/337

Grassa, R.L., Gallo, I., Landro, N. (2021). Learn class hierarchy using convolutional neural networks. *Appl. Intell.*, *51*(10), 6622–6632, https://doi.org/10.1007/S10489-020-02103-6 Retrieved from https://doi.org/10.1007/s10489-020-02103-6

Huo, X., Sun, G., Tian, S., Wang, Y., Yu, L., Long, J., . . . Li, A. (2024). Hifuse: Hierarchical multi-scale feature fusion network for medical image classification. *Biomed. Signal Process. Control.*, *87*(Part A), 105534, https://doi.org/10.1016/J.BSPC.2023.105534 Retrieved from https://doi.org/10.1016/j.bspc.2023.105534

Iwano, K., Shibuya, S., Kagiwada, S., Iyatomi, H. (2024). Hierarchical object detection and recognition framework for practical plant disease diagnosis. *CoRR*, *abs/2407.17906*, , https://doi.org/10.48550/ARXIV.2407.17906 Retrieved from https://doi.org/10.48550/arXiv.2407.17906 2407.17906

Jain, N., Bartz, C., Bredow, T., Metzenthin, E., Otholt, J., Krestel, R. (2020). Semantic analysis of cultural heritage data: Aligning paintings and descriptions in art-historic collections. A.D. Bimbo et al. (Eds.), *Pattern recognition. ICPR international workshops and challenges - virtual event, january 10-15, 2021, proceedings, part III* (Vol. 12663, pp. 517–530). Springer. Retrieved from https://doi.org/10.1007/978-3-030-68796-0_37

28

Jocher, G., Chaurasia, A., Qiu, J. (2023). *Ultralytics yolo* [Software]. Retrieved from https://ultralytics.com

Kalhagen, E.S., Olsen, O.L., Goodwin, M., Gupta, A. (2022). Hierarchical object detection applied to fish species. *Nordic Machine Intelligence (NMI)*, *2*(1), 1–15, https://doi.org/10.5617/nmi.9452 Retrieved from https://hdl.handle.net/11250/3042042

Kiritchenko, S., Matwin, S., Nock, R., Famili, A.F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. *Advances in artificial intelligence, 19th conference of the canadian society for computational studies of intelligence, canadian AI 2006, québec city, québec, canada, june 7-9, 2006, proceedings.* Retrieved from https://doi.org/10.1007/11766247_34

Kobayashi, T. (2021). Group softmax loss with discriminative feature grouping. *IEEE winter conference on applications of computer vision, WACV 2021, waikoloa, hi, usa, january 3-8, 2021* (pp. 2614–2623). IEEE. Retrieved from https://doi.org/10.1109/WACV48630.2021.00266

Kosmopoulos, A., Partalas, I., Gaussier, É., Paliouras, G., Androutsopoulos, I. (2015). Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Min. Knowl. Discov.*, *29*(3), 820–865, https://doi.org/10.1007/S10618-014-0382-X Retrieved from https://doi.org/10.1007/s10618-014-0382-x

Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., ... Yang, J. (2020). Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems 33: Annual conference on neural information processing systems 2020, neurips 2020, december 6-12, 2020, virtual.* Retrieved from https://proceedings.neurips.cc/paper/2020/hash/f0bda020d2470f2e74990a07a607ebd9-Abstract.html

Mayouf, M., & de Saint-Cyr, F.D. (2022). GH-CNN: A new CNN for coherent hierarchical classification. E. Pimenidis, P. Angelov, C. Jayne, A. Papaleonidas, & M. Aydin (Eds.), *Artificial neural networks and machine learning - ICANN 2022 - 31st international conference on artificial neural networks, bristol, uk, september 6-9, 2022, proceedings, part IV* (Vol. 13532, pp. 669–681). Springer. Retrieved from https://doi.org/10.1007/978-3-031-15937-4_56

Muller, B.R., & Smith, W.A.P. (2020). A hierarchical loss for semantic segmentation. G.M. Farinella, P. Radeva, & J. Braz (Eds.), *Proceedings of the 15th international joint conference on computer vision, imaging and computer graphics theory and applications, VISIGRAPP 2020, volume 4: Visapp, valletta, malta, february 27-29, 2020* (pp. 260–267). SCITEPRESS. Retrieved from

https://doi.org/10.5220/0008946002600267

Neudecker, C. (2022). Cultural heritage as data: Digital curation and artificial intelligence in libraries. A. Paschke, G. Rehm, C. Neudecker, & L. Pintscher (Eds.), *Proceedings of the third conference on digital curation technologies (qurator 2022), berlin, germany, sept. 19th-23rd, 2022* (Vol. 3234). CEUR-WS.org. Retrieved from https://ceur-ws.org/Vol-3234/paper2.pdf

Ojwang, G., Ogutu, J., Said, M., Ojwala, M., Kifugo, S., Verones, F., . . . Olff, H. (2023, March 18). An integrated hierarchical classification and machine learning approach for mapping land use and land cover in complex social-ecological systems. *Frontiers in Remote Sensing*, *4*, , https://doi.org/10.3389/frsen.2023.1188635 (Publisher Copyright: Copyright © 2024 Ojwang, Ogutu, Said, Ojwala, Kifugo, Verones, Graae, Buitenwerf and Olff.)

Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. *2017 IEEE conference on computer vision and pattern recognition, CVPR 2017, honolulu, hi, usa, july 21-26, 2017* (pp. 6517–6525). IEEE Computer Society. Retrieved from https://doi.org/10.1109/CVPR.2017.690

Riehl, K., Neunteufel, M., Hemberg, M. (2023). Hierarchical confusion matrix for classification performance evaluation. *CoRR*, *abs/2306.09461*, , https://doi.org/10.48550/ARXIV.2306.09461 Retrieved from https://doi.org/10.48550/arXiv.2306.09461 2306.09461

Roy, D., Panda, P., Roy, K. (2020). Tree-cnn: A hierarchical deep convolutional neural network for incremental learning. *Neural Networks*, *121*, 148–160, https://doi.org/10.1016/J.NEUNET.2019.09.010 Retrieved from https://doi.org/10.1016/j.neunet.2019.09.010

Seo, Y., & Shin, K. (2019). Hierarchical convolutional neural networks for fashion image classification. *Expert Syst. Appl.*, *116*, 328–339, https://doi.org/10.1016/J.ESWA.2018.09.022 Retrieved from https://doi.org/10.1016/j.eswa.2018.09.022

Taoufiq, S., Nagy, B., Benedek, C. (2020). Hierarchynet: Hierarchical cnn-based urban building classification. *Remote. Sens.*, *12*(22), 3794, https://doi.org/10.3390/RS12223794 Retrieved from https://doi.org/10.3390/rs12223794

Terven, J.R., Córdova-Esparza, D., Romero-González, J. (2023). A comprehensive review of YOLO architectures in computer vision: From yolov1 to yolov8 and YOLO-NAS. *Mach. Learn. Knowl. Extr.*, *5*(4), 1680–1716, https://doi.org/10.3390/MAKE5040083 Retrieved from https://doi.org/10.3390/make5040083

Tharwat, A. (2020). Classification assessment methods. *Applied Computing and Informatics*, , Retrieved from https://api.semanticscholar.org/CorpusID:59212480

Usmani, M.A., Mahmood, S., Elmadany, Y., Azeem, A.Z., Zualkernan, I. (2025). Hierarchical yolo with real-time text recognition for uae traffic signs. *Authorea Preprints*, ,

Wang, B., & Barbu, A. (2023). Hierarchical classification for large-scale learning. *Electronics*, *12*(22), , https://doi.org/10.3390/electronics12224646 Retrieved from https://www.mdpi.com/2079-9292/12/22/4646

Wei, Y., Tran, S.N., Xu, S., Kang, B.H., Springer, M. (2020). Deep learning for retail product recognition: Challenges and techniques. *Comput. Intell. Neurosci.*, *2020*, 8875910:1–8875910:23, https://doi.org/10.1155/2020/8875910 Retrieved from https://doi.org/10.1155/2020/8875910

Wen, Y., Zhang, K., Li, Z., Qiao, Y. (2016). A discriminative feature learning approach for deep face recognition. B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision - ECCV 2016 - 14th european conference, amsterdam, the netherlands, october 11-14, 2016, proceedings, part VII* (Vol. 9911, pp. 499–515). Springer. Retrieved from https://doi.org/10.1007/978-3-319-46478-7_31

Yang, J., Gao, M., Kuang, K., Ni, B., She, Y., Xie, D., Chen, C. (2020). Hierarchical classification of pulmonary lesions: A large-scale radio-pathomics study. A.L. Martel et al. (Eds.), *Medical image computing and computer assisted intervention - MICCAI 2020 - 23rd international conference, lima, peru, october 4-8, 2020, proceedings, part VI* (Vol. 12266, pp. 497–507). Springer. Retrieved from https://doi.org/10.1007/978-3-030-59725-2_48

Yaseen, M. (2024). What is yolov8: An in-depth exploration of the internal features of the next-generation object detector. *CoRR*, *abs/2408.15857*, , https://doi.org/10.48550/ARXIV.2408.15857 Retrieved from https://doi.org/10.48550/arXiv.2408.15857 2408.15857

Yu, Z., Nguyên, T.D., Gal, Y., Ju, L., Chandra, S.S., Zhang, L., ... Ge, Z. (2022). Skin lesion recognition with class-hierarchy regularized hyperbolic embeddings. L. Wang, Q. Dou, P.T. Fletcher, S. Speidel, & S. Li (Eds.), *Medical image computing and computer assisted intervention - MICCAI 2022 - 25th international conference, singapore, september 18-22, 2022, proceedings, part III* (Vol. 13433, pp. 594–603). Springer. Retrieved from https://doi.org/10.1007/978-3-031-16437-8_57
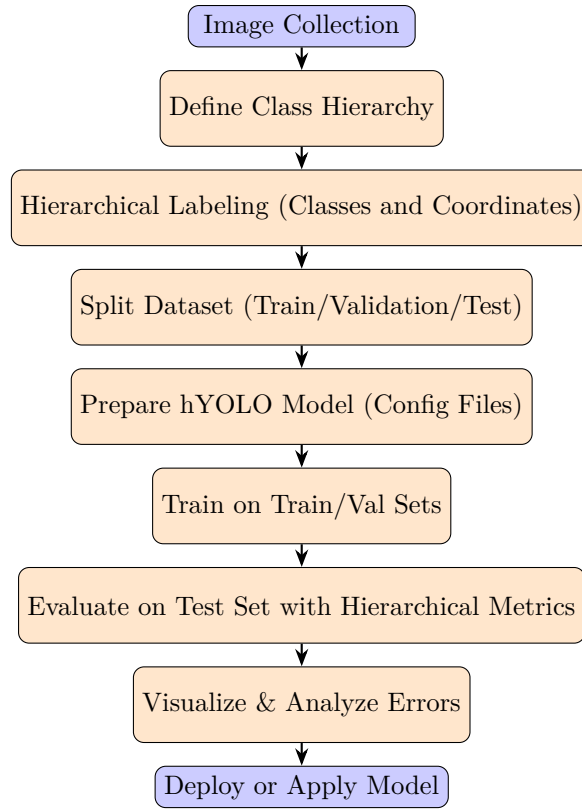
Zhang, Z., Zhang, X., Peng, C., Xue, X., Sun, J. (2018). Exfuse: Enhancing feature fusion for semantic segmentation. V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision - ECCV 2018 - 15th european conference, munich, germany, september 8-14, 2018, proceedings, part X* (Vol. 11214, pp. 273–288). Springer. Retrieved from https://doi.org/10.1007/978-3-030-01249-6_17

Zheng, Q., Luo, Z., Guo, M., Wang, X., Wu, R., Meng, Q., Dong, G. (2025). HGO-YOLO: advancing anomaly behavior detection with hierarchical features and lightweight optimized detection. *CoRR*, *abs/2503.07371*, , https://doi.org/10.48550/ARXIV.2503.07371 Retrieved from https://doi.org/10.48550/arXiv.2503.07371 2503.07371

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D. (2020). Distance-iou loss: Faster and better learning for bounding box regression. *The thirty-fourth AAAI conference on artificial intelligence, AAAI 2020, the thirty-second innovative applications of artificial intelligence conference, IAAI 2020, the tenth AAAI symposium on educational advances in artificial intelligence, EAAI 2020, new york, ny, usa, february 7-12, 2020* (pp. 12993–13000). AAAI Press. Retrieved from https://doi.org/10.1609/aaai.v34i07.6999

Zhu, X., & Bain, M. (2017). B-CNN: branch convolutional neural network for hierarchical classification. *CoRR*, *abs/1709.09890*, , Retrieved from http://arxiv.org/abs/1709.09890 1709.09890

Zunaed, M., & Fattah, S.A. (2022). A novel hierarchical-classification-block based convolutional neural network for source camera model identification. *CoRR*, *abs/2212.04161*, , https://doi.org/10.48550/ARXIV.2212.04161 Retrieved from https://doi.org/10.48550/arXiv.2212.04161 2212.04161

Appendix A   Hierarchical Model Training Workflow

The process to train a hierarchical object detection and classification model involves several well-defined stages. The workflow (Figure A1) begins with data preparation and annotation structured according to a class hierarchy, followed by dataset partitioning and model configuration. Subsequent steps focus on training, evaluation using hierarchical metrics, error analysis, and model deployment for practical applications. The following outlines each step in detail:

- **Image Collection:** This initial step involves gathering a comprehensive set of images relevant to the problem domain. Both the quality and diversity of the collected images directly impact the robustness and generalizability of the model developed later. Additionally, it is important to include images with multiple objects present, potentially overlapping, to better simulate real-world scenarios. To further enhance the dataset, various data augmentation techniques such as rotation, scaling, flipping, and color adjustments should be applied to increase variability and improve the model's ability to generalize across different conditions.
- **Define Class Hierarchy:** A meaningful and structured taxonomy of classes has to be established to represent the relationships between different categories, ideally based on visual similarities or semantic relevance. This hierarchical organization helps the model utilize class dependencies effectively during training.
- **Hierarchical Labeling (Classes and Coordinates):** In this step, each image is annotated not only with its multi-level class label but also with precise object coordinates such as bounding boxes. Hierarchical labeling ensures that labels respect the defined class structure, enhancing the model's ability to detect and classify objects at multiple levels.
- **Split Dataset (Train/Validation/Test):** The dataset is partitioned into training, validation, and testing subsets. This division is important for unbiased model training, hyperparameter tuning, and performance evaluation on unseen data.
- **Prepare hYOLO Model (Config Files, see Section C):** This step involves configuring the hierarchical YOLO (hYOLO) model to handle multi-level object detection and classification. It typically requires modifying configuration files, such as the default YAML, to specify the hierarchical class structure, including the names and number of classes at each level. In addition, essential parameters set during this step include the number of training epochs, commonly 300, which controls the total iterations over the dataset; the batch size, which determines how many images are processed simultaneously in each training iteration; and the input image size, often set to 1280 pixels, which affects detection precision and computational requirements. Overall, this step ensures that the model is properly configured to effectively learn both object localization and classification across the multiple hierarchical levels represented in the data.
- **Train on Train/Val Sets:** The model is trained using the training dataset, with the validation set used to monitor progress and prevent overfitting. Iterative updates are made to the model parameters to minimize the loss function and improve predictive accuracy.

- **Evaluate on Test Set with Hierarchical Metrics:** The trained model's performance is assessed on the test set using hierarchical metrics. This step provides an objective measure of how well the model generalizes to new, unseen data.
- **Visualize & Analyze Errors:** Results and misclassifications are visualized to identify patterns of errors or weaknesses in the model. This analysis helps identify areas for improvement, such as enhancing data augmentation, fine-tuning the model parameters, or adjusting the training strategy.
- **Deploy or Apply Model:** Finally, the validated model is deployed in a real-world environment or integrated into an application. This deployment may involve embedding the model on edge devices, such as Raspberry Pi, NVIDIA Jetson Nano, or other compact embedded systems, enabling real-time object classification directly within the operational environment. This step also includes ongoing monitoring of the model's performance post-deployment and planning for future updates or refinements as necessary.

Image Collection
↓
Define Class Hierarchy
↓
Hierarchical Labeling (Classes and Coordinates)
↓
Split Dataset (Train/Validation/Test)
↓
Prepare hYOLO Model (Config Files)
↓
Train on Train/Val Sets
↓
Evaluate on Test Set with Hierarchical Metrics
↓
Visualize & Analyze Errors
↓
Deploy or Apply Model

Fig. A1: Workflow diagram for hierarchical object detection and classification using the hYOLO framework.

Appendix B   Training Dataset Samples

Fig. B2: An example image from the dataset.

Fig. B3: Different expositions of the same object.

Appendix C   Listings

Listing 1: Changes in the default parameters in *YOLOv8* in cfg/default.yaml when adding hierarchy

```
# Ultralytics YOLO , GPL -3.0 license
# Default training settings and hyperparameters for medium -augmentation COCO
    training

task: detect   # YOLO task, i.e. detect, segment, classify, pose
mode: train  # YOLO mode, i.e. train, val, predict, export, track, benchmark
#hier_yolo: True
#hierarchy_names:
hier_depth: 5
calc_TP_FN_FP: False
calc_set_metric: False
get_hier_paths: False
calc_TP_FP_conf: False
dependency_loss: True


# Train settings
---------------------------------------------------------------------------
model:   # path to model file, i.e. yolov8n.pt, yolov8n.yaml
data:   # path to data file, i.e. coco128.yaml
epochs: 300  # number of epochs to train for
patience: 30  # epochs to wait for no observable improvement for early
    stopping of training
batch: 1  # number of images per batch (-1 for AutoBatch)
imgsz: 1280  # size of input images as integer or w,h
save: True  # save train checkpoints and predict results
save_period: 10 # Save checkpoint every x epochs (disabled if < 1)
cache: disk  # True/ram, disk or False. Use cache for data loading
device: cpu # device to run on, i.e. cuda device=0 or device=0,1,2,3 or
    device=cpu
workers: 8  # number of worker threads for data loading (per RANK if DDP)
project:  runs/hier_yolo # project name
name:   #train_100class_hier_v2 # experiment name, results saved to
    'project/name' directory
exist_ok: False  # whether to overwrite existing experiment
pretrained: False  # whether to use a pretrained model
optimizer: SGD  # optimizer to use, choices=['SGD', 'Adam', 'AdamW',
    'RMSProp']
verbose: True  # whether to print verbose output
seed: 0  # random seed for reproducibility
deterministic: True  # whether to enable deterministic mode
single_cls: False  # train multi-class data as single-class
image_weights: False  # use weighted image selection for training
rect: False  # support rectangular training if mode='train', support
    rectangular evaluation if mode='val'
cos_lr: False  # use cosine learning rate scheduler
close_mosaic: 0  # disable mosaic augmentation for final 10 epochs
resume: False  # resume training from last checkpoint
amp: False  # Automatic Mixed Precision (AMP) training, choices=[True,
    False], True runs AMP check
# Segmentation
overlap_mask: True  # masks should overlap during training (segment train
    only)
mask_ratio: 4  # mask downsample ratio (segment train only)
# Classification
dropout: 0.0  # use dropout regularization (classify train only)


# Val/Test settings
---------------------------------------------------------------------------
val: True  # validate/test during training
```

```
split: val  # dataset split to use for validation, i.e. 'val', 'test' or
    'train'
save_json: False  # save results to JSON file
save_hybrid: False  # save hybrid version of labels (labels + additional
    predictions)
conf:  # object confidence threshold for detection (default 0.25 predict,
    0.001 val)
iou: 0.7  # intersection over union (IoU) threshold for NMS

max_det: 300  # maximum number of detections per image
half: False  # use half precision (FP16)
dnn: False  # use OpenCV DNN for ONNX inference
plots: True  # save plots during train/val

# Prediction settings
--------------------------------------------------------------------------------
source:  # source directory for images or videos
show: False  # show results if possible
save_txt: False  # save results as .txt file
save_conf: False  # save results with confidence scores
save_crop: False  # save cropped images with results
hide_labels: False  # hide labels
hide_conf: False  # hide confidence scores
vid_stride: 1  # video frame-rate stride
line_thickness: 3  # bounding box thickness (pixels)
visualize: False  # visualize model features
augment: False  # apply image augmentation to prediction sources
agnostic_nms: False  # class-agnostic NMS
classes:  # filter results by class, i.e. class=0, or class=[0,2,3]
retina_masks: False  # use high-resolution segmentation masks
boxes: True  # Show boxes in segmentation predictions

# Export settings
--------------------------------------------------------------------------------
format: torchscript  # format to export to
keras: False  # use Keras
optimize: False  # TorchScript: optimize for mobile
int8: False  # CoreML/TF INT8 quantization
dynamic: False  # ONNX/TF/TensorRT: dynamic axes
simplify: False  # ONNX: simplify model
opset:  # ONNX: opset version (optional)
workspace: 4  # TensorRT: workspace size (GB)
nms: False  # CoreML: add NMS

# Hyperparameters
--------------------------------------------------------------------------------
lr0: 0.01  # initial learning rate (i.e. SGD=1E-2, Adam=1E-3)
lrf: 0.01  # final learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 7.5  # box loss gain
cls: 2.0  # cls loss gain (scale with pixels)
dfl: 1.5  # dfl loss gain
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
label_smoothing: 0.0  # label smoothing (fraction)
nbs: 64  # nominal batch size
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
```

```
fliplr: 0.5  # image flip left-right (probability)
mosaic: 0.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
copy_paste: 0.0  # segment copy-paste (probability)
```