YOLO26-Based Pollen Detection and Classification in Brightfield Microscopy Images

Jinnan Wang (s3997928)

Supervisors:
Dr. Lu Cao & Nemi Dorst

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

June 29, 2026

Abstract

Several wind-pollinated plant species release allergenic pollen during flowering, which can cause hay fever and other allergic symptoms in sensitized individuals. However, traditional manual identification and quantification of pollen using brightfield microscopy are time-consuming and require expert knowledge. In recent years, deep learning methods have been applied to automated pollen detection tasks. This paper presents the performance evaluation of the latest YOLO26 model released in 2026 for the task of microscopic pollen image detection and compares it with the YOLOv8 model used in our previous experiments. Apart from the baseline model comparisons, several training strategies and architectural modifications are also explored in this study, namely freeze backbone, class weights, regmax16, and an additional P2 detection layer head. To assess the stability and robustness of the models, five-fold cross-validation has been employed. The results of the experiments demonstrate that YOLO26 and the training strategies and architecture modifications based on it have indeed enhanced the performance of detection. By evaluating YOLO26 against YOLOv8 and testing several training and architectural modifications, this study provides evidence to assess which model configurations are most suitable for automated pollen detection in light microscopy images.

Contents

1 Introduction

A significant proportion of the European population suffers from pollen allergies and it is known that pollen is the primary aeroallergen responsible for allergic rhinitis [4]. Quality of life is strongly affected by this type of allergic disease, while the health costs and loss of productivity are also very significant [2]. Climate change has also been associated with longer pollen seasons, increased pollen production, and the spread of allergenic species to new areas. Therefore, accurate monitoring and classification of airborne pollen is very important for allergy forecasting, environmental monitoring, and public health management.

Until now, pollen analysis has been carried out manually by palynology experts using light microscopy. This technique is slow, laborious, and likely to involve human bias in classification. An important problem is discriminating between different pollen species that are morphologically very similar. For example, some Cupressaceae species, like *Callitropsis nootkatensis* and *Chamaecyparis lawsoniana*, are hard to differentiate even for very experienced analysts using only a light microscope [7]. These limitations of light microscopy call for a highly accurate method that can enhance both efficiency and taxonomic resolution.

Recent progress in deep learning has greatly boosted the accuracy of automated image analysis. In particular, object detection models based on convolutional neural networks have demonstrated superior performance in applications to biological and medical imaging. Previous research on the pollen dataset used in this thesis involved testing several object detection models, including RetinaNet, DINO (DETR-Swin), and YOLOv8 [18]. Of these methods, YOLOv8 achieved the best overall performance and demonstrated strong potential for automated pollen detection and classification.

In 2026, YOLO26 [9] was released as a new member of the YOLO family, with many architectural and training changes compared to its predecessors. However, the efficacy of YOLO26 for microscopic pollen detection has not been assessed. Therefore, it is still unknown whether YOLO26 can outperform YOLOv8 on this dataset and if additional training strategies and architectural modifications can further enhance its performance.

This thesis aims to evaluate the performance of YOLO26 in the detection and classification of three allergenic pollen species: *Betula pendula*, *Callitropsis nootkatensis*, and *Chamaecyparis lawsoniana*. It also compares training strategies and architectural modifications directly with YOLOv8s, Freeze Backbone, Class Weights, RegMax16, and an additional P2 detection layer head. To assess the stability and robustness of the proposed models across different data distributions, five-fold cross-validation is used.

The thesis addresses the following research questions:

- Research Question 1: Does YOLO26 outperform YOLOv8 for pollen detection?
- Research Question 2: Can training strategies and architectural modifications further improve YOLO26 performance?

2 Background

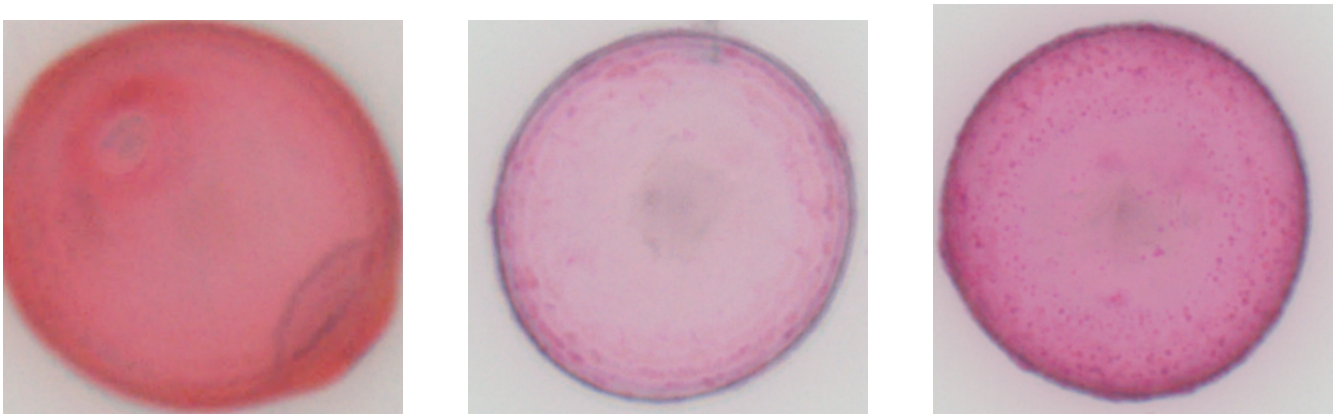
2.1 Pollen Dataset and Light Microscopy Imaging

This dataset was created for and used in a previous study [18] on automated detection and classification of pollen. It comprises microscopic images of three allergenic tree pollen species that are found in the Netherlands:

- *B. pendula*
- *C. nootkatensis*
- *C. lawsoniana*

For better morphological visibility under light microscopy, the samples were stained with safranine. The slides were imaged using a Zeiss Axioscan 7 brightfield microscopy scanner. A 10x prescan was first acquired, after which the images were captured at 40x magnification with three colour channels. 20-layer z-stacks were captured with a fixed step size to ensure that pollen grains at different depths were represented.

To obtain training images that can be used for deep learning, the z-stacks were projected to two-dimensional images. In the final models, minimum projection was used as it maintains the sharpest grain boundaries and surface textures. The original 2056 by 2464 pixel images were divided into 1024 by 1024 pixel tiles to reduce computational requirements. Five-fold cross-validation was used to assess model robustness and stability across different data distributions. Figure 1 gives examples of the three pollen species in the dataset.



(a) *B.pendula* (b) *C.nootkatensis* (c) *C.lawsoniana*

Figure 1: Representative microscopy images of the three pollen species in the dataset.

2.2 YOLO Architecture

Object detection models are designed to identify and classify objects within images. In light microscopy image analysis, this is useful because multiple pollen grains may appear in the same image. Previous study on this pollen dataset compared several one-stage object detection models,

including RetinaNet, DINO (DETR-Swin), and YOLOv8. YOLOv8 achieved the best overall performance in that study [18], which motivates its use as the baseline model. Since YOLO26 is a newer YOLO model, this thesis focuses on evaluating whether it can improve pollen detection performance compared with YOLOv8.

2.2.1 YOLOv8

YOLOv8 [8] is a real-time object detection model released by Ultralytics in 2023. It has gained broad popularity as an object detection framework because of its strong detection performance and easy implementation. The model can be trained and deployed with just a few lines of code using the Ultralytics package. As a one-stage detection model, it simplifies the process of identifying objects by performing localization and classification in a single step [15].

Compared to earlier versions of YOLO, YOLOv8 comes with a few architectural changes. In the first place, it uses a detection mechanism without anchors, getting rid of the predefined anchor boxes and making the process of training easier. Secondly, YOLOv8 uses a decoupled detection head that separates classification from bounding box regression into different branches so that each task can be optimized independently. The other addition in the model is the C2f module that enhances feature extraction and gradient flow [22].

The general structure of YOLOv8 is made up of three main parts: a backbone, a neck, and a detection head as illustrated in Figure 2. The backbone is responsible for feature extraction hierarchically from the input image, while the Path Aggregation Network (PANet) neck allows feature combination at different scales for better multi-scale object detection [12]. Finally, the decoupled detection head does the prediction of object classes and bounding boxes at different levels of features [17].

The YOLOv8 comes in five model sizes: Nano (n), Small (s), Medium (m), Large (l), and Extra Large (x) [8]. All these versions have the same architecture but vary in the number of parameters and computational complexity (GFLOPs). In general, the smaller models have faster inference times and lower memory consumption than the larger ones, which generally have higher detection accuracies but require more computational resources. This allows the user to choose an appropriate model based on the hardware available and the requirements of the application. In this work, the YOLOv8s was chosen as the reference model since it represents a good trade-off between accuracy and computational efficiency.

2.2.2 YOLO26

YOLO26 [9] is the most recent object detection model presented by Ultralytics in 2026. Building on other versions of YOLO, YOLO26 comes with a few enhancements for improving detection accuracy, lower inference latency, and make model deployment easier. In the official introduction, it is claimed that YOLO26 has higher COCO mAP scores than YOLOv8 and keeps low inference latency, which means it has a better balance between speed and accuracy. Figure 3 presents a comparison of the performance of YOLO26 and YOLOv8 on the COCO benchmark across different model sizes[22]. In every model size, YOLO26 is better than YOLOv8 in terms of both mAP50-95 scores and inference time.

YOLO26 has the same backbone–neck–head architecture as YOLOv8, but it comes with some critical updates. First, it switches to an end-to-end detection framework, so Non-Maximum Sup-

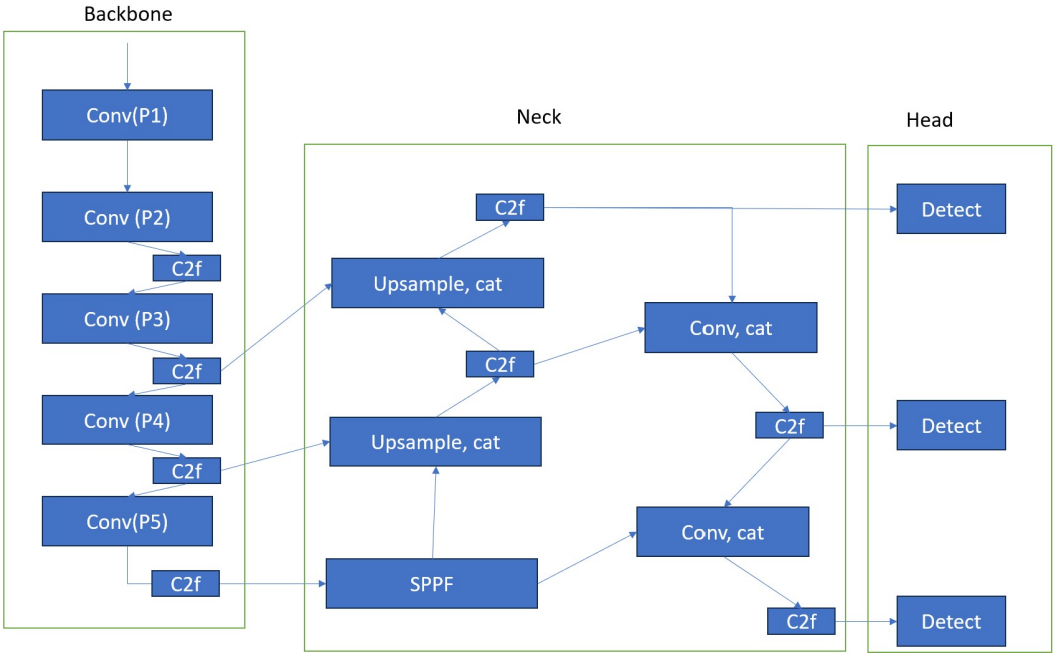


Figure 2: Simplified architecture of YOLOv8. The model consists of a backbone with convolutional and C2f modules, a neck for multi-scale feature fusion, and a detection head for object classification and localization.

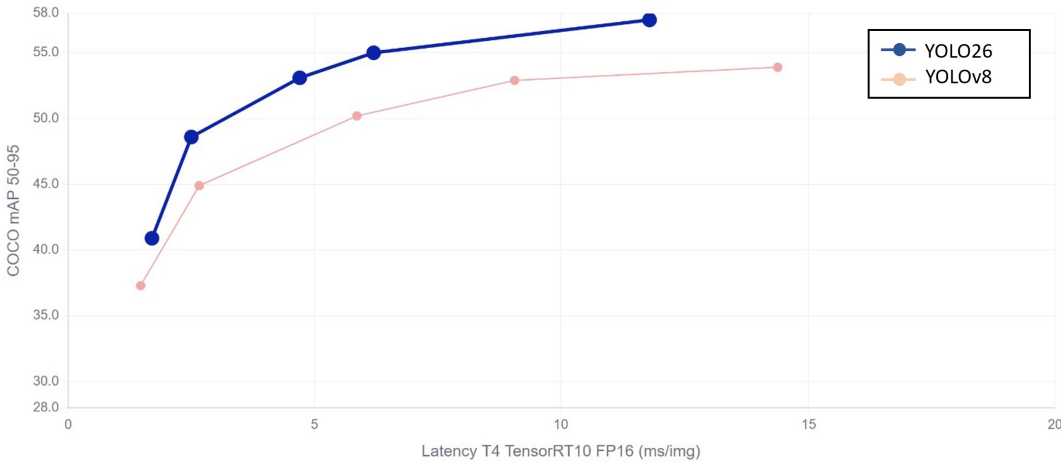


Figure 3: Comparison of YOLO26 and YOLOv8 on the COCO benchmark [22]. The blue and orange curves represent YOLO26 and YOLOv8, respectively. YOLO26 achieves higher mAP50-95 scores while maintaining lower inference latency across different model sizes (n, s, m, l, x).

pression (NMS) is no longer necessary during inference[6]. This change makes the detection pipeline more straightforward and eases the post-processing burden. Second, it gets rid of Distribution Focal Loss (DFL) and moves to a simpler regression mechanism[6]. With NMS and DFL eliminated, the computational complexity is reduced, significantly simplifying model export and deployment across various hardware platforms. The result is that YOLO26 can deliver faster inference and lower latency on devices with limited resource[22].

In addition, YOLO26 also brings about several training enhancements, including Small-Target-Aware Label Assignment (STAL), Progressive Loss (ProgLoss), and the MuSGD optimizer[9]. MuSGD is an optimizer that was recently proposed as a hybrid approach that combines the Muon optimizer with the standard Stochastic Gradient Descent (SGD). MuSGD works quite differently from traditional optimizers like AdamW or SGD as it uses a hybrid update strategy based on different types of parameters. Together, all these features make YOLO26 a particularly good fit for real-time applications and edge computing.

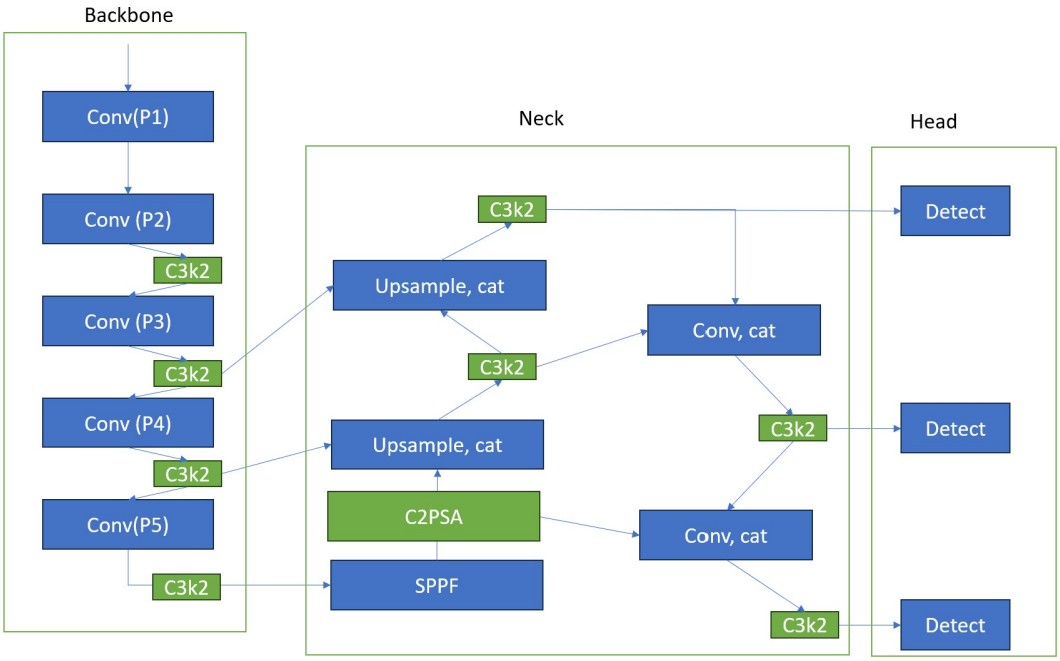


Figure 4: Simplified YOLO26 architecture. Green blocks highlight the main differences from YOLOv8: the C2f modules are replaced by C3k2 modules and the introduction of a C2PSA module in the neck.

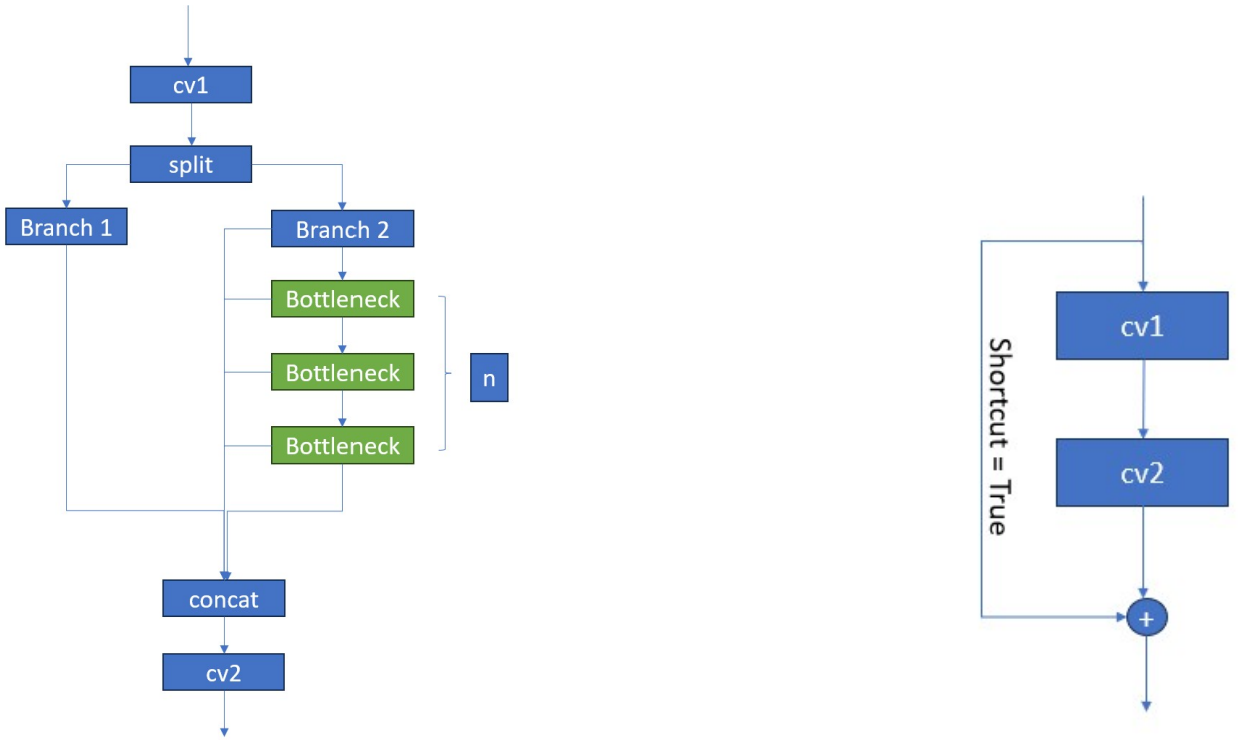
2.2.3 Comparison of C2f, C3k2, and C2PSA Modules

The main architectural difference in terms of feature extraction modules between YOLOv8 and YOLO26 is the following. In the architecture, YOLOv8 uses C2f mainly, while YOLO26 brings in C3k2 and C2PSA to enhance feature representation and detection performance.

C2f: The C2f module is one of the main components of YOLOv8, built based on the Cross Stage Partial (CSP) architecture[23]. As illustrated in Figure 5a, the input feature map first goes through a convolution layer (cv1) and then is split into two branches. One branch keeps the original feature information, while the other one goes through a few Bottleneck blocks to get deeper features.

The output of both branches is concatenated and the final convolution layer (cv2) fuses it. The proposed structure boosts gradient flow, feature reuse, at maintained computational cost. Hence, C2f is a module with high feature extraction ability and has been widely used in real-time object detection models.

In the C2f module, the Bottleneck module is a residual block. As shown in Figure 5b, it is composed of two convolutional layers and a shortcut connection. The first layer (cv1) decreases the feature dimension and extracts intermediate representations, while the second layer (cv2) further processes the features and restores the output dimension. When the shortcut parameter is set to True and the input and output channels are the same, the shortcut connection gets activated and the input feature map is added straight to the output via a residual connection [5]. Such a design resolves the problem of gradient vanishing and proves beneficial for training deeper neural networks.



(a) C2f module

(b) Bottleneck module

Figure 5: Architectures of the C2f and Bottleneck modules used in YOLOv8. The C2f module designs feature extraction and fusion, while the Bottleneck module utilizes a residual connection.

C3k2 [9]: YOLO26 replaces C2f modules with C3k2 modules. As illustrated in Figure 6, C3k2 follows an architecture based on CSP that divides the feature map into two branches, akin to C2f. One of the branches is a direct bypass to the intermediate process, while the other goes through some feature extraction blocks before the process of feature fusion. These blocks are Bottleneck block, C3k block, Bottleneck block followed by PSABlock. The choice between the feature extraction blocks is governed by two boolean parameters.

In the attention mechanism, the feature extraction branch includes a Bottleneck block and then a PSABlock (Position-Sensitive Attention Block). The Bottleneck block extracts local features from the first convolution operations, while the PSABlock builds long-range spatial relationships

by applying multi-head self-attention and a feed-forward network. The proposed method allows for more context-based image analysis by combining local convolutional features with global attention information, thereby enabling the network to focus more on context of an image and capture subtle morphological differences between visually similar pollen species.

When the C3k block is enabled, the module extends the standard Bottleneck design by introducing configurable convolution kernel sizes. This way, we can try out different kernel sizes for one particular task and find the configuration that gives the best detection performance on a certain dataset.

With Bottleneck block enabled, C3k2 will perform like the C2f module in YOLOv8, having two convolutional layers and a residual connection. Moreover, C3k2 can be set up to use larger kernel convolutions or attention blocks. This way, the model can capture richer spatial information and learn more complex visual patterns — all while holding onto efficient inference speed. YOLO26 introduces both the C3k2 and C2PSA modules to improve feature extraction and attention-based feature enhancement [9] [16], as illustrated in Figure 6.

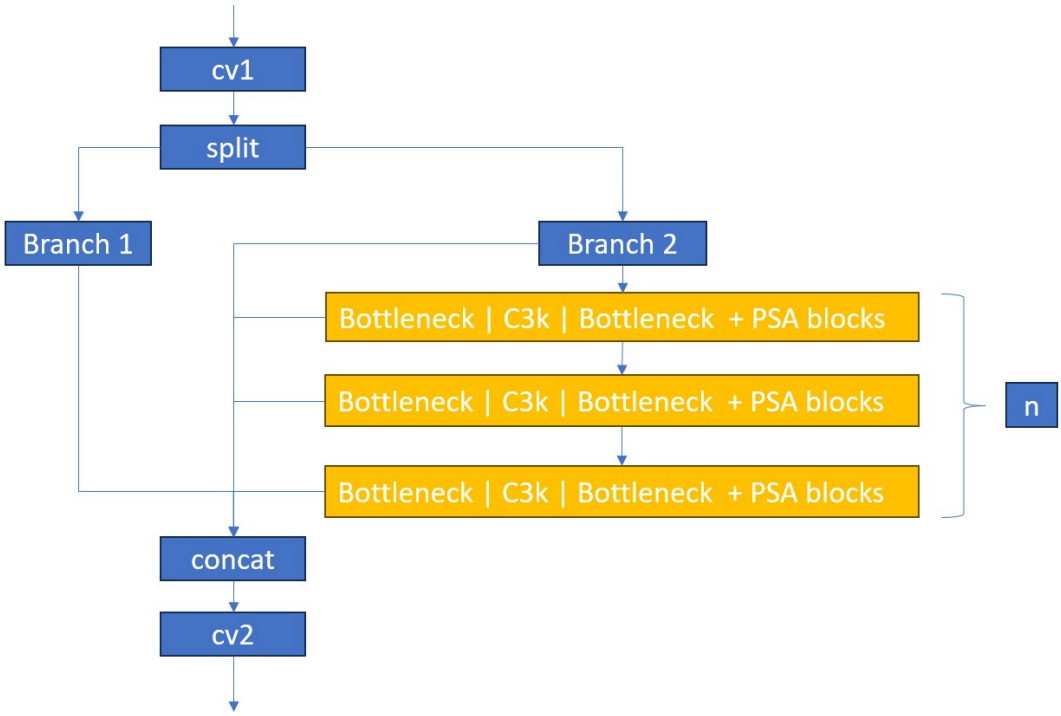


Figure 6: Architecture of the C3k2 module used in YOLO26. The feature extraction branch consists of n blocks, where each block can be implemented as a Bottleneck block, a C3k block, or a Bottleneck block followed by a PSABlock. The outputs from all branches are concatenated and fused through a final convolution layer (cv2).

C2PSA [6]: Another part brought in YOLO26 is the C2PSA module, which also brings attention mechanisms into the network. As shown in Figure 7, the whole structure of C2PSA is the same as the CSP-based design used in C2f, but the Bottleneck blocks are replaced with PSABlocks. The input feature map goes through a convolution layer (cv1) first and then is split into two branches. One branch does not have any further processing and keeps the original feature information, while the other branch goes through a series of (n) PSABlocks. The two branch

outputs are finally concatenated and fused by a last convolution layer (cv2).

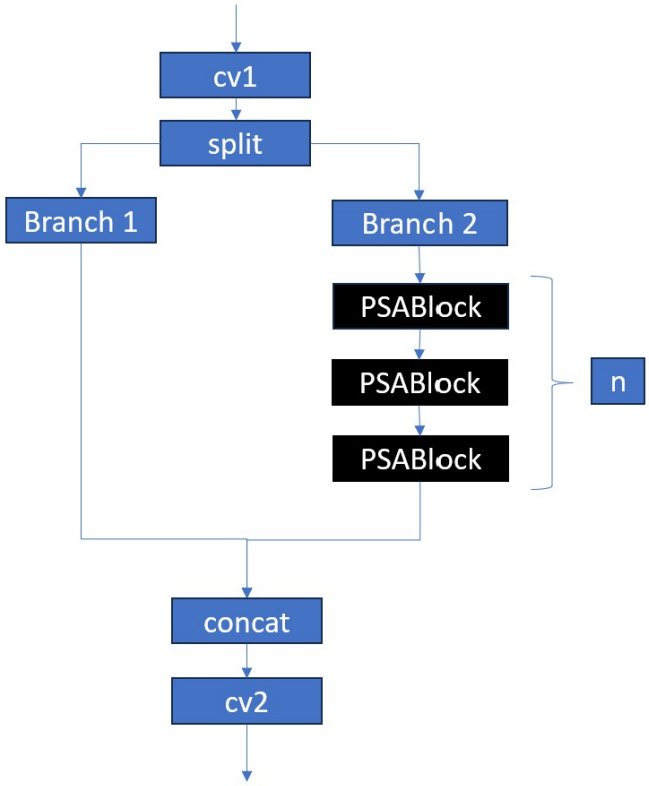


Figure 7: Architecture of the C2PSA module in YOLO26. The input feature map is divided into a bypass branch and an attention branch containing n PSABlocks. The two branches are concatenated and fused through a final convolution layer.

C2PSA is an attention-based feature refinement module. This enables the module to refine high-level features while keeping the efficiency of computation.

The PSABlock used in C2PSA is the same as that used in the C3k2 module. It is a self-attention mechanism and feed-forward network like a standard Transformer encoder block. They are connected through residual connections. The difference is that it does not include Layer Normalization layers, unlike a standard Transformer encoder. With more PSABlocks included, it helps in enhancing global feature modeling and allows the network to capture long-range spatial dependencies before the features are passed to the neck and detection head.

This can be useful for pollen detection tasks because the fine differences in texture and morphology often demand the model to observe the spatial relationships among the entire pollen grain instead of depending on single local features.

Dual-Head End-to-End Detection [9]: Another major difference between YOLOv8 and YOLO26 is the addition of a dual-head detection architecture. In contrast to YOLOv8, which trains with a single dense detection head and Non-Maximum Suppression (NMS), YOLO26 trains with both a one-to-many branch and a one-to-one branch.

The one-to-many branch follows the conventional YOLO detection paradigm. Each ground-truth object can be assigned to multiple positive samples through Task-Aligned Learning (TAL), providing

richer supervision and improving training stability. During inference, duplicate predictions are removed using NMS.

In contrast, the one-to-one branch assigns exactly one prediction to each ground-truth object. This branch is designed for end-to-end object detection and does not use NMS at inference time. Because each object has only one corresponding prediction, the post-processing stage can be much simplified, which reduces inference time and improves deployment efficiency.

As shown in Figure 8, the two branches adopt different assignment strategies. During training, both branches are optimized simultaneously. The one-to-many branch gives dense supervision, while the one-to-one branch learns the final inference behavior. This design makes it possible for YOLO26 to support both high-accuracy NMS-based prediction and efficient NMS-free end-to-end detection [3].

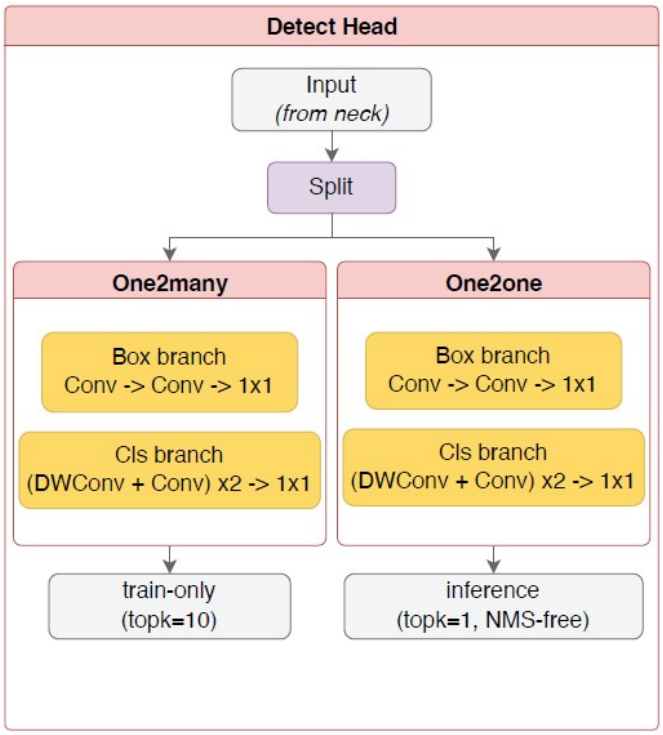


Figure 8: Dual-head detection design in YOLO26. The one-to-many branch provides dense predictions and uses NMS, while the one-to-one branch supports end-to-end NMS-free inference [9].

In this study, the one-to-many branch was used for model validation and testing because no hardware constraints were considered. It is also able to reach higher detection performance on the pollen dataset.

3 Related Work

3.1 Pollen Classification and Detection

Early studies of automated pollen monitoring mainly focused on traditional machine learning methods using handcrafted features. To distinguish morphologically similar species, researchers extracted features such as texture, shape, and geometric properties, including Gray-Level Co-occurrence Matrix (GLCM) and Local Binary Patterns (LBP) features [10]. Li et al. [10] showed that a hierarchical machine-learning strategy achieved 94.5% accuracy for Urticaceae pollen classification. Earlier work by Marcos et al. [13] also used texture features together with Fisher's Discriminant Analysis and K-Nearest Neighbour classification, reaching 95% accuracy. However, the performance of these approaches depends strongly on the quality of manually selected features, and feature-based methods may be less robust when the background is unclear.

The field shifted towards deep learning, especially Convolutional Neural Networks (CNNs). Polling et al. [14] used VGG16 and MobileNet models for allergenic pollen monitoring and processed three-dimensional pollen information using z-stack images. Later, Li et al. [10] reported that ResNet50 achieved an accuracy of 99.4% for Urticaceae pollen classification. However, much of this previous work focused on classifying pollen grains. In contrast, object detection is needed when multiple pollen grains have to be localized and classified within microscopy images. A recent study compared one-stage models, including RetinaNet, DINO, and YOLOv8s [18]. Among these models, YOLOv8s achieved the best overall performance for identifying and classifying three common allergenic pollen species in the Netherlands: *B. pendula*, *C. nootkatensis*, and *C. lawsoniana*, making it a strong baseline for the present study.

In the study, all models were trained with minimum projection images that were created from light microscopy z-stacks [18]. The validation results indicated that YOLOv8s got the best performance among the models. After hyperparameter optimization, the model attained a validation precision of 96.3% and an mAP50 score of 98.6%. Furthermore, 5-fold cross-validation revealed a mean mAP50 score of 79.3% with a standard deviation of 7.48% and a mean F1 score of 74.7% with a standard deviation of 6.1% on the test set, which shows relatively stable performance over different train and validation splits.

3.2 Research Gap

The authors have stated that YOLOv8 performed better than RetinaNet and DINO on their pollen dataset, but those experiments were conducted before YOLO26 was ever released. Thus, they have not explored the performance of newer YOLO architectures on this task. YOLO26 comes with several updates over YOLOv8, such as the C3k2 module, C2PSA module, and the MuSGD optimization strategy [16]. According to the official YOLO26 paper, these changes should be able to give a more balanced performance in terms of detection accuracy and speed of inference, but we are not sure if they will work for microscopic pollen detection.

Moreover, previous optimization experiments primarily concentrated on hyperparameter tuning, including learning rate, weight decay, optimizer, and model size. It is still unknown if YOLO26 can beat YOLOv8 on this dataset and if additional training strategies and architecture modifications can further enhance detection performance. To fill this research gap, this thesis studies the performance of YOLO26 on the same pollen dataset and compares its results with the YOLOv8s benchmark.

4 Approach

4.1 Dataset

The pollen dataset described in Section 2.1 was used in this study [18]. The dataset contains annotated instances of three pollen species, namely *B. pendula*, *C. nootkatensis*, and *C. lawsoniana*. To maintain consistency with the previous study, the dataset was divided into a training-validation set and an independent test set. The test set was reserved for final evaluation and was not used during the processes of training, validation, or hyperparameter optimization.

Each fold contains a predefined training and validation split following an approximately 80/20 ratio and the same class distribution across the three pollen species. Table 1 summarizes the distribution of pollen instances in each fold and the test set.

Due to the vast number of experiments and huge computational expenses on training models, a two-stage evaluation strategy was used. In the model selection phase, all candidate models were trained and evaluated in Fold 0 only. The best-performing models were chosen for further research. In the final evaluation stage, the top three models were evaluated with the complete 5-fold cross-validation method. Performance metrics reported for the final models correspond to the average results across all five folds.

As shown in Table 1, the five folds exhibit highly consistent class distributions, in which *B. pendula* represents approximately 45% of all instances, followed by *C. nootkatensis* (approximately 33%) and *C. lawsoniana* (approximately 22%). In contrast, the test set has a different class distribution, where *C. nootkatensis* becomes the majority class, accounting for 51.5% of all pollen instances. This difference was not deliberately introduced; the independent test set was created from independent scans to mimic real-world use [18].

Table 1: Class distribution across the five folds and the independent test set [18].

Fold	**Split**	**Total**	***B. pendula***	***C. nootkatensis***	***C. lawsoniana***
Fold 0	Train	4549	2071 (45.5%)	1481 (32.6%)	997 (21.9%)
	Val	1144	512 (44.8%)	379 (33.1%)	253 (22.1%)
Fold 1	Train	4551	2057 (45.2%)	1488 (32.7%)	1006 (22.1%)
	Val	1142	526 (46.1%)	372 (32.6%)	244 (21.4%)
Fold 2	Train	4524	2040 (45.1%)	1483 (32.8%)	1001 (22.1%)
	Val	1169	543 (46.5%)	377 (32.3%)	249 (21.3%)
Fold 3	Train	4568	2073 (45.4%)	1500 (32.8%)	995 (21.8%)
	Val	1125	510 (45.3%)	360 (32.0%)	255 (22.7%)
Fold 4	Train	4580	2091 (45.7%)	1488 (32.5%)	1001 (21.9%)
	Val	1113	492 (44.2%)	372 (33.4%)	249 (22.4%)
Test Set	Test	1736	516 (29.7%)	894 (51.5%)	326 (18.8%)

4.2 Hyperparameter Optimization

Hyperparameters are the preset values of training, which are fixed before the training of the model and influence the optimization process directly. According to the documentation of Ultralytics

YOLO, there is a wide range of hyperparameters we can optimize. Hyperparameter optimization is a step to enhance the performance of the model. Ultralytics states that hyperparameter optimization uses the genetic algorithm, which iteratively generates new hyperparameter combinations through mutation and selection, and then evaluates them using a process similar to model training and validation[19].

In this thesis, hyperparameter optimization was performed for the YOLO26s baseline model. The optimization was conducted on Fold 0 to reduce computational cost while still providing a consistent training and validation setting for model selection.

Six hyperparameters were selected for tuning: the initial learning rate (`lr0`), final learning rate factor (`lrf`), momentum (`momentum`), weight decay (`weight_decay`), bounding box loss weight (`box`), and classification loss weight (`cls`). These parameters were chosen because they directly affect convergence behaviour, regularization, localization accuracy, and classification performance. The search space used during optimization is summarized in Table 2. The optimized hyperparameters obtained from this process were then used as the default training configuration for all YOLO26s-based experiments.

Table 2: Hyperparameter search space used for YOLO26s optimization[19].

Hyperparameter	Range	Description
`lr0`	[1e-5, 1e-2]	Initial learning rate at the start of training. Lower values provide more stable training but slower convergence.
`lrf`	[0.01, 1.0]	Final learning rate factor as a fraction of `lr0`. Controls how much the learning rate decreases during training.
`momentum`	[0.7, 0.98]	Momentum factor. Higher values help maintain consistent gradient direction and can speed up convergence.
`weight_decay`	[0.0, 0.001]	L2 regularization factor to prevent overfitting. Larger values enforce stronger regularization.
`box`	[1.0, 20.0]	Bounding box loss weight in the total loss function. Balances box regression and classification performance.
`cls`	[0.1, 4.0]	Classification loss weight in the total loss function. Higher values emphasize correct class prediction.

4.3 Baseline Models

4.3.1 YOLOv8s

The YOLOv8s was chosen as the baseline model because it achieved the best overall performance in the previous study on the same dataset. Using the same model architecture makes it easy to compare directly between the previous results and the improvements in this thesis.

All experiments were conducted on 2 NVIDIA RTX3080 GPUs with 8GB memory. The YOLOv8s models were initialized with pretrained weights, trained on the COCO dataset. To cut the overall experiment time, we did not run a full hyperparameter optimization procedure for YOLOv8s. Instead, we used the same initial learning rate and weight decay values as in the previous study [18].

A simple grid search was conducted to find the best training settings for the present experimental setup. It involved the batch size and optimizer shown in Table 3. The combination that achieved the highest validation performance on Fold 0 was selected and then used for the other models.

Other training parameters were set according to the optimized settings described in Section Hyperparameter Optimization. The final YOLOv8s configuration was used as the baseline for comparison with the YOLO26s-based models from training strategies and architectural modifications.

Table 3: Grid search configuration

Parameter	Candidates
Batch size	[4, 8, 16]
Optimizer	[SGD, AdamW, MuSGD]

4.3.2 YOLO26s

YOLO26s was chosen to make a fair comparison with YOLOv8s. Both models have approximately the same number of parameters. YOLO26s has about 9.5 million parameters, while YOLOv8s has about 11.2 million parameters.

The model was initialized using COCO-pretrained weights from Ultralytics package. The optimized hyperparameters described in Section 4.2 were used. To ensure a consistent experimental setup, the same batch size selected for YOLOv8s was adopted for YOLO26s. MuSGD was used as the optimizer, following the recommendation of the YOLO26, as it was designed to improve training behaviour compared with conventional SGD.

Unless otherwise stated, the training schedule and other training settings were kept consistent with those for YOLOv8s training. Table 4 summarizes the training settings. The resulting YOLO26s configuration also served as the baseline for evaluating the proposed training strategies and architectural modifications.

Table 4: YOLO26s training configuration.

Parameter	**Value**
Model	YOLO26s
Image Size	1024
Epochs	100
Batch Size	16
Optimizer	MuSGD
Pretrained	True
Initial Learning Rate (`lr0`)	Optimized
Final Learning Rate Factor (`lrf`)	Optimized
Momentum	Optimized
Weight Decay	Optimized
Box Loss Weight (`box`)	Optimized
Classification Loss Weight (`cls`)	Optimized

4.4 Training Strategies

4.4.1 Freeze backbone

Backbone freezing is a training technique in which selected layers of a pretrained network are prevented from updating during training. According to the Ultralytics documentation, freezing layers allows the model to retain previously learned visual features while focusing training on task-specific components [21]. This approach can improve training stability and reduce overfitting, particularly when working with relatively small datasets.

Since the backbone is initialized with COCO-pretrained weights, it has already learned general visual representations such as edges, shapes, textures, and object patterns from a large scale dataset. By freezing the backbone during the early stage of training, the model can focus on updating the detection head, which is responsible for adapting these generic features to the pollen detection task. This may accelerate the adaptation of task-specific parameters while preserving the useful feature representations learned during pretraining.

In this study, a two-stage training strategy was adopted. During the first 40 epochs, the backbone network was frozen while the detection head remained trainable. After 40 epochs, the backbone was unfrozen and the entire network was trained for the last 60 epochs. The total training duration remained 100 epochs, consistent with the baseline configuration.

4.4.2 Class Weights

Class weighting is a technique used to address class imbalance during training. According to the Ultralytics documentation, class weights can be incorporated into the loss function. This encourages the model to pay more attention to minority classes during training.

The class weights were computed using an inverse frequency weighting. For each class, the weight is computed as the ratio between the number of instances in the majority class and the number of instances in that class:

$$w_i = \frac{N_{\max}}{N_i}$$

where w_i is the weight for class i, N_i is the number of training instances of class i, and $N_{\max}$ is the number of instances in the most frequent class. With this formulation, the majority class receives a weight of 1, while less frequent classes receive larger weights.

For handling class imbalance, a class-weighted Binary Cross-Entropy (BCE) loss is used for the classification component, following the custom trainer method provided by Ultralytics [20]. Although the task contains multiple pollen classes, YOLO treats classification as independent binary predictions for each class. The classification loss for each class i is defined as:

$$L_{cls,i} = -\left[w_i y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

where y_i is the ground-truth binary label for class i, $\hat{y}_i$ is the predicted probability after sigmoid activation. The weight w_i is the positive weight and applied only to the positive term, following the `pos_weight` formulation used in `BCEWithLogitsLoss` [1]. When an underrepresented class is present in an image, this class becomes the positive target. If the model assigns a low probability to this positive class, the positive term of the loss function increases and is further amplified by w_i. As

a result, incorrect or poorly predicted minority class objects receive a larger penalty, encouraging the model to learn these classes more effectively. The total classification loss is obtained by aggregating the losses across all N classes:

$$L_{cls} = \sum_{i=0}^{N-1} L_{cls,i}$$

The resulting class weights used in this study are summarized in Table 5.

Table 5: Class frequencies and corresponding class weights used during training.

Pollen Species	Instances	Class Weight
B. pendula	2071	1.00
C. nootkatensis	1481	1.40
C. lawsoniana	997	2.08

4.4.3 Combine Freeze backbone and Class Weights

To investigate whether combining the two strategies can further improve model performance, both methods were applied simultaneously. The training process followed the same two-stage schedule described in Section 4.4.1. During the first 40 epochs, the backbone network remained frozen while the detection head was trained. After 40 epochs, the backbone was unfrozen and the entire network was trained jointly for the remaining 60 epochs.

Class weights were incorporated into the classification loss throughout the entire training process, including both the frozen and unfrozen stages. The class weights were calculated using the inverse frequency weighting described in Section 4.4.2. All other training parameters remained identical to the YOLO26s baseline configuration.

4.5 Architecture Modifications

4.5.1 Regmax16 for Bounding Box Regression (Regmax16)

YOLOv8 employs Distribution Focal Loss (DFL) for bounding box regression [8]. Instead of directly predicting the coordinates of a bounding box, DFL predicts a discrete probability distribution over multiple bins. The final regression value is obtained by calculating the expectation of the predicted distribution:

$$d = \sum_{i=0}^{K-1} i \cdot \text{softmax}(z)_i$$

where K denotes the number of bins, which is from the RegMax parameter and z represents the predicted logits. In YOLOv8, the default value is `reg_max=16`, meaning that 16 bins are used to calculate the distribution of each box coordinate.

In contrast, YOLO26 removes DFL and adopts direct box regression with `reg_max=1`, which is a default value in the configuration [6]. In this way, each box coordinate is represented by a single regression value rather than a probability distribution. As a result, this reduces the number

of output channels in the detection head with fewer model parameters and lower computational cost. The removal of DFL is one of the factors contributing to the smaller model size of YOLO26 compared with YOLOv8.

Since this thesis focuses on detection performance rather than computational efficiency, an alternative YOLO26s configuration was evaluated by restoring `reg_max=16`. This modification reintroduces DFL into the detection head while keeping all other training settings unchanged. The experiment was designed to investigate whether the finer localization capability provided by DFL can improve pollen detection performance.

4.5.2 Adding P2 Layer Head

Multi-scale feature maps are used in object detection to improve detection across object sizes [11]. The default YOLOv8s and YOLO26s architectures employ three detection layers based on feature maps P3, P4, and P5. Their spatial resolutions correspond to 1/8, 1/16, and 1/32 of the input image, respectively [8] [9]. These detection layers are designed to capture objects at different scales. P3 primarily focuses on small objects, P4 on middle size objects, and P5 on large objects.

However, repeated downsampling in the backbone reduces the spatial resolution of feature maps. For some objects, important fine-grained details may be lost before reaching the detection head. This issue may be particularly relevant for pollen classification, as distinguishing between species can depend on subtle differences in shape, boundary structures, and surface textures.

To preserve higher-resolution spatial information, an additional P2 detection layer was introduced. Figure 9 illustrates the modified YOLO26 architecture with the added P2 detection layer. The P2 feature map operates at a stride of 4 pixels and has a spatial resolution of 256×256 for an input image size of 1024×1024. Compared with P3, the P2 feature map contains more information, which allows the detector to better represent small objects and fine morphological details.

4.5.3 Combine Regmax16 and Adding P2 Layer Head

Similar to the combined training strategy, both architectural modifications were applied simultaneously to investigate whether combining the two methods can further improve the model performance.

4.6 5-Fold Cross Validation

Based on the Fold 0 evaluation, the three best-performing models were selected for further assessment using 5-fold cross-validation. As YOLOv8s was the baseline model in this study, it was also included in the cross-validation evaluation if it was not in the top 3 based on mAP50-95. Consequently, all selected models were evaluated using the complete 5-fold cross-validation procedure.

Each model was trained and evaluated on all five folds of the dataset. The final cross-validation performance was calculated as the average of the evaluation metrics obtained across the five folds. This approach provides a more robust estimate of model performance. After cross-validation, the selected models were evaluated on the independent test set to assess their generalization ability.

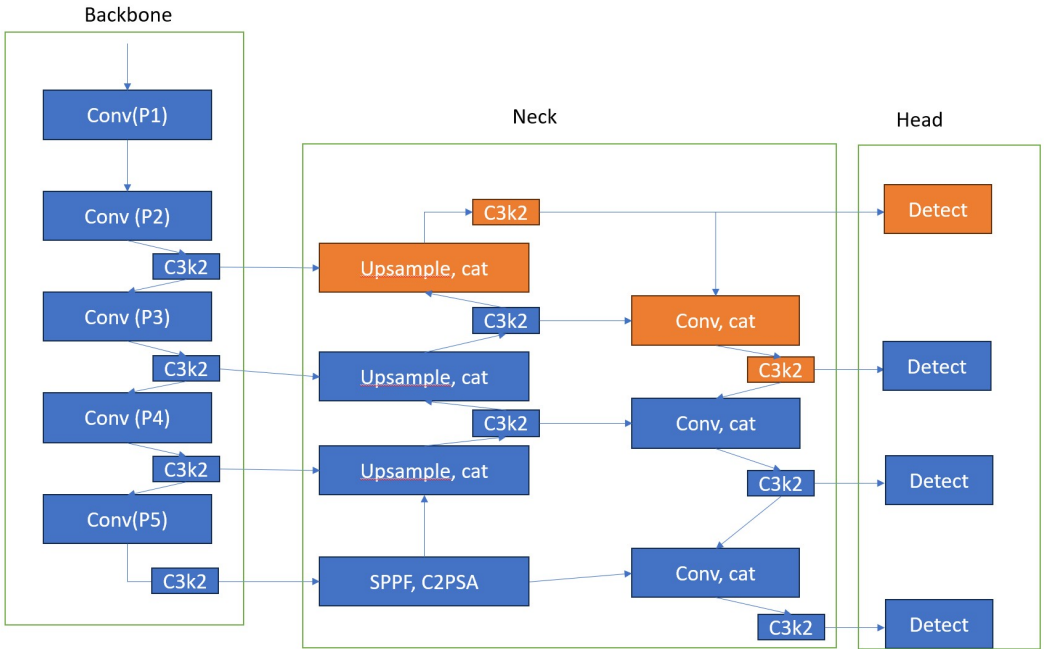


Figure 9: YOLO26 with an additional P2 detection head. The orange boxes indicate the new modules used to generate the P2 feature map and the corresponding detection head.

5 Results

In this section, I will present the performance of the different models in training stage and their evaluation results on the validation and test sets.

5.1 Hyperparameter Optimization Results

During the hyperparameter tuning process, to reduce the computational cost of the search process, each iteration was trained for 40 epochs rather than the full 100 epochs used in the final experiments. A total of 50 trials were conducted on Fold 0, with each trial corresponding to a candidate hyperparameter configuration. The full search required approximately 3.5 days of training time. Model fitness is defined as the mAP50-95 score, which is the default optimization objective in Ultralytics. As shown in Figure 10a, although fluctuations are observed between individual iterations, the smoothed fitness curve shows a gradual improvement from approximately 0.832 at the beginning of the search to around 0.846 after 50 iterations. This indicates that the algorithm is searching for better hyperparameter combinations during the search process, resulting in improved model performance.

Figure 10b presents the distribution of candidate hyperparameter values explored during optimization. The cross marker in each subplot indicates the best hyperparameter value selected by the optimization process. The highest fitness score was achieved by the hyperparameter configuration in Table 6.

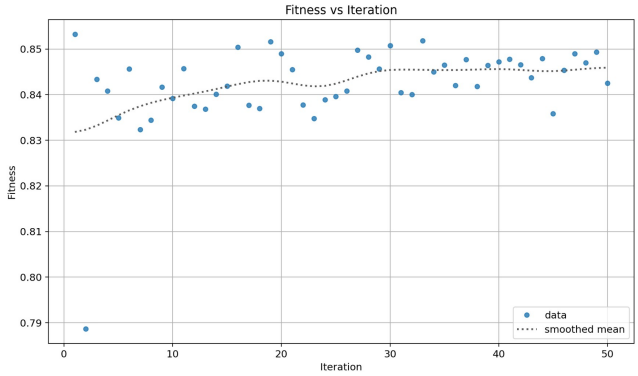
Table 6: Best hyperparameter configuration obtained through optimization.

Hyperparameter	Value
`lr0`	0.01
`lrf`	0.01
`weight_decay`	0.0005
`momentum`	0.937
`box`	7.5
`cls`	0.5

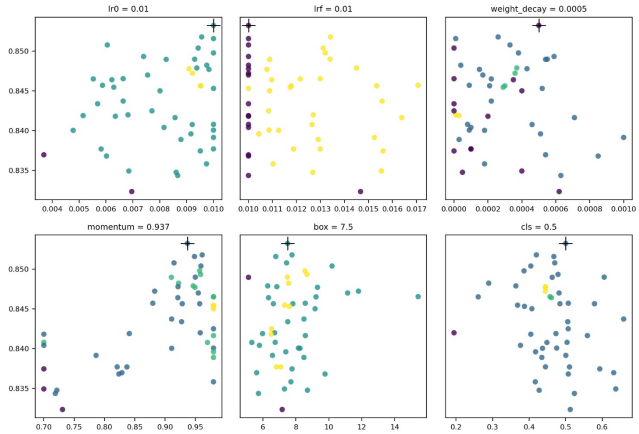
5.2 Training Results

Figure 11 shows the training and validation loss curves for the YOLOv8s baseline and the Freeze Backbone model. The first row is YOLOv8s model, while the second row shows the Freeze Backbone model. The box loss, classification loss, and DFL loss are shown from left to right. For YOLOv8s, both the training and validation losses decreased steadily during training. Similar loss patterns were observed for most of the other models. These models completed the full 100 training epochs without signs of overfitting.

In contrast, the Freeze Backbone model exhibited a different training behaviour. During the first 40 epochs, when the backbone remained frozen, the validation losses gradually increased while the training losses continued to decrease, indicating overfitting in the frozen stage. After the backbone was unfrozen, the validation losses decreased again. Both models employing the Freeze Backbone strategy triggered early stopping and therefore did not complete the full 100 training epochs.



(a) Fitness scores during hyperparameter optimization.



(b) Distribution of explored hyperparameter values.

Figure 10: Hyperparameter optimization results. (a) Fitness scores obtained during the search process. (b) Distribution of candidate hyperparameter values, where the cross marker indicates the best hyperparameter value.

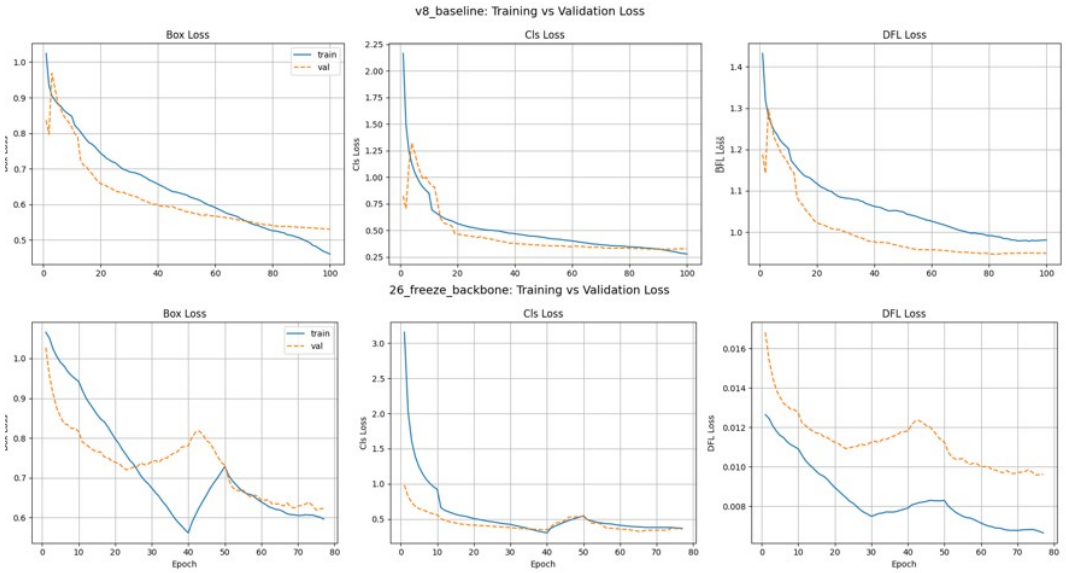


Figure 11: Training and validation loss curves for the YOLOv8 baseline (top row) and the Freeze Backbone model (bottom row).

Figure 12 shows the validation mAP50-95 curves of all eight models during training. Since Ultralytics saves the best model checkpoint according to the highest mAP50-95 score, this metric is used to compare model performance. As we can observe from the plot that most models show a increase in mAP50-95 in training stage. Among all evaluated models, the Class Weight model achieves the highest mAP50-95 value, reaching approximately 0.85. This result suggests that addressing class imbalance is beneficial for the pollen detection task.

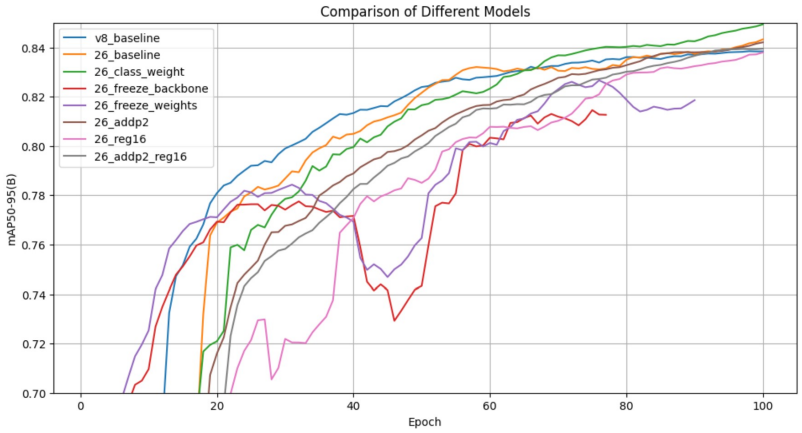


Figure 12: mAP50-95 curves for all evaluated models during training. The best model checkpoint is selected according to the highest mAP50-95 score.

5.3 Validation and Model Selecting

The best checkpoint of each model was evaluated on the Fold 0 validation set. For each model, Precision, Recall, mAP50, and mAP50-95 were used for performance comparison. To visualize the results clear, Figure 13 plots mAP50-95 against F1 score for all evaluated models. The x-axis represents mAP50-95 and the y-axis is F1 score. Models that are closer to the upper-right corner achieve both higher detection accuracy and a better balance between precision and recall.

As shown in Figure 13, the Class Weight model reaches the best overall performance, its mAP50-95 and F1 score are highest among all models. Based on the validation mAP50-95 results, the top three models are selected for further evaluation: Class Weight, YOLO26 baseline, and RegMax16.

5.4 Cross-Validation and Test Set Performance

Table 7 summarizes the performance of the test set of the selected models after 5-fold cross-validation. To make the relationship between detection performance and stability easier to interpret, Figure 14 summarizes the test set performance of the selected models as a scatter plot. The x-axis represents the mean mAP50-95 obtained from the five folds, while the y-axis represents the corresponding standard deviation (SD). Models closer to the lower-right corner have both higher performance and greater stability.

The YOLO26 baseline model achieves the highest mean mAP50-95. However, the Class Weight model achieves a comparable mean performance since it exhibits the lowest standard deviation. This

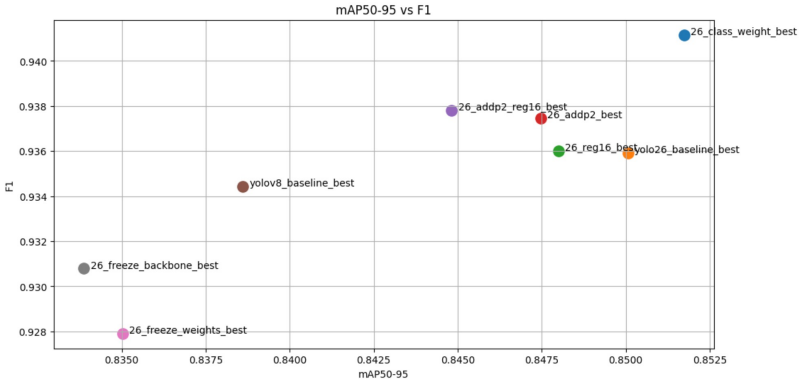


Figure 13: Comparison on Fold 0 validation set. The x-axis is mAP50-95 and the y-axis is F1 score. Models closer to the upper-right corner achieve better overall performance.

Table 7: Cross-validation performance on the test set. Values are reported as mean ± standard deviation across five folds.

Model	Precision	Recall	F1	mAP50	mAP50–95
YOLO26 baseline	0.899 ± 0.009	0.850 ± 0.015	0.867 ± 0.015	0.912 ± 0.009	0.741 ± 0.009
Class Weight	0.905 ± 0.013	0.826 ± 0.016	0.854 ± 0.013	0.915 ± 0.007	0.737 ± 0.004
RegMax16	0.826 ± 0.081	0.811 ± 0.009	0.792 ± 0.060	0.881 ± 0.030	0.708 ± 0.027
YOLOv8 baseline	0.885 ± 0.012	0.821 ± 0.012	0.841 ± 0.012	0.905 ± 0.011	0.716 ± 0.006

result indicates that the Class Weight strategy provides more robust and consistent performance across folds. In contrast, the RegMax16 model shows both lower mean performance and higher variability.

5.5 Case Analysis

Figure 15 presents an example from the test set using the best YOLO26s baseline model. The left image shows the detection results and the right image visualizes the corresponding activation heatmap with all ground-truth annotations.

The model successfully detected and classified the fully visible *B. pendula* pollen grain. In addition, the model also detected a pollen grain that was only partially visible in the image. The detection confidence for this partial object is relatively high (0.59). Since each slide in the dataset contains a single pollen species, this partially visible object is expected to be *B. pendula*. This suggests that the model had learned discriminative pollen features rather than only relying on complete object appearances. This observation is also supported by the heatmap visualization. Only one bounding box is shown in the heatmap because the partially visible pollen grain was not included in the ground-truth annotation. Strong activation can be observed around the partially visible pollen, indicating that the model focused on meaningful pollen features when predicting an object.

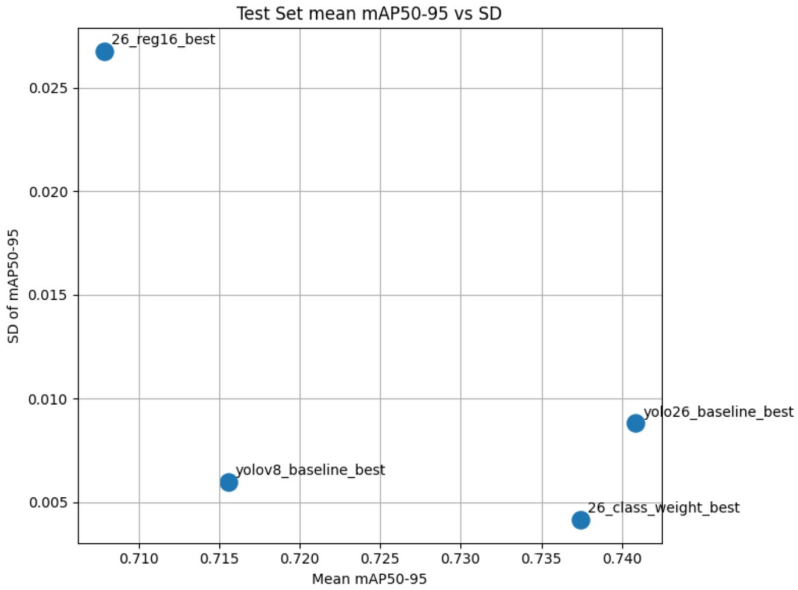


Figure 14: Mean test-set mAP50–95 versus standard deviation across the five folds.

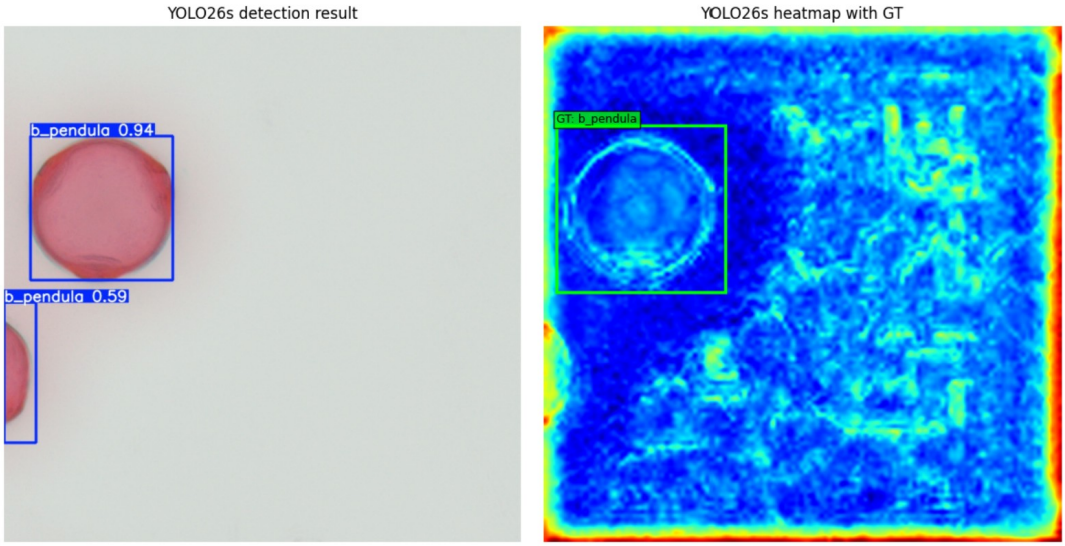


Figure 15: Qualitative analysis of a test set image using the best YOLO26s baseline model. (a) Prediction result. (b) Heatmap with ground-truth annotation. (Only the fully visible pollen grain is annotated as ground truth.)

6 Discussion

In this study, the newly released YOLO26s model was used and compared with the YOLOv8s model that was used in previous research for the detection and classification of three pollen species. The results show that YOLO26s outperformed YOLOv8s. The best-performing model, the Class Weight model, achieved a mean mAP50-95 of 73.7% (SD = 0.4%), a mean mAP50 of 91.5% (SD = 0.7%) and a mean F1 score of 85.4% (SD = 1.3%) on the test set. These results are substantially higher than the previously reported YOLOv8s performance, which achieved a mean mAP50 of 79.3% (SD = 7.48%) and a mean F1 score of 74.7% (SD = 6.1%) on the test set.

These findings indicate that YOLO26s not only outperforms YOLOv8s for pollen detection, but YOLO26 variants can further improve upon the YOLO26s model. The performance improvement is likely related to the architectural upgrades in YOLO26. The C3k2 and C2PSA modules provide stronger feature extraction capabilities than the C2f module used in YOLOv8. Unlike conventional convolutional operations, these modules incorporate attention mechanisms that allow the model to capture both local features and global contextual information. This characteristic is important for pollen detection. The visual differences between pollen species are often subtle, and successful classification may require the model to consider the overall morphology of the pollen grain rather than local texture patterns.

The Class Weight model had the best performance among all models. This indicates that class balance has a significant impact on model performance. Its impact is even greater than increasing the number of model parameters. It is generally believed that increasing the number of parameters can improve model capability. However, in this study, both the P2 model and the RegMax16 model increased the number of parameters, but neither achieved better performance than the Class Weight model. The Freeze Backbone models showed the worst performance. One possible explanation is that the pretrained weights were obtained from the COCO dataset. The samples in the COCO dataset are very different from pollen images. As a result, the features learned by the model in the early stage could not be effectively transferred to the pollen detection task.

There are also some limitations that should be considered in this study. A total of eight different YOLO models were evaluated. Due to time constraints, hyperparameter tuning was only performed for the YOLO26s baseline model. The same optimized hyperparameters were then applied to all other models. As a result, this hyperparameter combination may not be the best for the other models. In addition, the class distribution of the test set is different from that in five folds. This may also be one of the reasons that the performance on the test set was lower than that on the validation set.

7 Further Research

Future research could further improve the performance of YOLO26s-based models. The results of this study show that YOLO26s-based models outperform YOLOv8s, but there are still several aspects that could be further explored.

First, as mentioned in the Discussion section, the training parameters used in this study were obtained from hyperparameter tuning. However, to save tuning time, hyperparameter tuning was only performed for the YOLO26s baseline model, and the same hyperparameters were then applied to other models. These parameters were not optimized separately for each model. Therefore, future work could perform individual hyperparameter tuning, such as for the best performance model in this study, which may further improve the results.

Second, the results show that the sample distribution of the dataset can affect the performance of models. In this study, the class distribution of the test set was different from that of the training set, and then a difference in performance was also observed between validation and test results. Future work could collect more samples to reduce the distribution gap between the training set and the test set. In addition, the dataset in this study is relatively clean, which makes the learning task easier for the model. To improve the practical value of the model, future studies could use more realistic datasets that contain more background noise or debris, so that the fine-tuned models can be applied to a wider range of real-world scenarios.

Finally, the case analysis shows that after learning sufficient pollen features, the model can recognize pollen grains that are not annotated in the dataset. However, because these pollen grains are not included in the ground-truth annotations, correct detections may be counted as false positives during evaluation. This may lead to an underestimation of the actual detection ability of the model. Therefore, future work should further check and improve the dataset annotations, so that the ground truth is more complete and the evaluation results can better reflect the true capability of the model.

8 Conclusion

Pollen is an important cause of seasonal allergic diseases, and accurate pollen monitoring is valuable for public health and allergy management. In this study, we compared the detection and classification performance of YOLOv8s and YOLO26s-based models for three allergenic pollen species in light microscopy images: *B. pendula*, *C. nootkatensis*, and *C. lawsoniana*. We further modified training strategies and model architectures to compare the performance of different YOLO26s variants. The results demonstrated that YOLO26s was better than YOLOv8s, and some YOLO26s variants performed even better, such as the Class Weight model.

These results indicate that YOLO26s is a promising model for automated pollen detection and classification, and that performance can be further enhanced with task-specific training strategies. The findings can be used as a basis to optimize pollen detection models. In addition, more models could be developed and tested based on the YOLO26s variants evaluated in this study.

References

[1] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. URL: https://docs.pytorch.org/assets/pytorch2-2.pdf, doi:10.1145/3620665.3640366.

[2] J. Bousquet, N. Khaltaev, A. A. Cruz, J. Denburg, W. J. Fokkens, A. Togias, T. Zuberbier, C. E. Baena-Cagnani, G. W. Canonica, C. van Weel, et al. Allergic rhinitis and its impact on asthma (aria) 2008 update: in collaboration with the world health organization, ga(2)len and allergen. *Allergy*, 63(Supplement 86):8–160, Apr. 2008. doi:10.1111/j.1398-9995.2007.01620.x.

[3] S. Chakrabarty. Yolo26: An analysis of nms-free end to end framework for real-time object detection, 2026. URL: https://arxiv.org/abs/2601.12882, arXiv:2601.12882.

[4] G. D'Amato, L. Cecchi, S. Bonini, C. Nunes, I. Annesi-Maesano, H. Behrendt, G. Liccardi, T. Popov, and P. van Cauwenberge. Allergenic pollen and pollen allergy in europe. *Allergy*, 62(9):976–990, Sept. 2007. doi:10.1111/j.1398-9995.2007.01393.x.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL: http://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[6] P. Hidayatullah and R. Tubagus. Yolo26: A comprehensive architecture overview and key improvements, 2026. URL: https://arxiv.org/abs/2602.14582, arXiv:2602.14582.

[7] I. Hordijk. The influence of background on machine learning-based pollen classification. Master's thesis, Leiden University, Leiden, The Netherlands, 2025.

[8] G. Jocher, A. Chaurasia, and J. Qiu. Ultralytics yolov8, 2023. URL: https://github.com/ultralytics/ultralytics.

[9] G. Jocher, J. Qiu, M. Liu, S. Lyu, F. C. Akyon, and M. E. Kalfaoglu. Ultralytics yolo26: Unified real-time end-to-end vision models, 2026. URL: https://arxiv.org/abs/2606.03748, arXiv:2606.03748.

[10] C. Li, M. Polling, L. Cao, B. Gravendeel, and F. J. Verbeek. Analysis of automatic image classification methods for urticaceae pollen classification. *Neurocomputing*, 522:181–193, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0925231222014308, doi:10.1016/j.neucom.2022.11.042.

[11] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2017. URL: https://arxiv.org/abs/1612.03144, arXiv:1612.03144.

[12] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018. URL: http://arxiv.org/abs/1803.01534, arXiv:1803.01534.

[13] J. V. Marcos, R. Nava, G. Cristóbal, R. Redondo, B. Escalante-Ramírez, G. Bueno, Óscar Déniz, A. González-Porto, C. Pardo, F. Chung, and T. Rodríguez. Automated pollen identification using microscopic imaging and texture analysis. *Micron*, 68:36–46, 2015. URL: https://www.sciencedirect.com/science/article/pii/S096843281400167X, doi:10.1016/j.micron.2014.09.002.

[14] M. Polling, C. Li, L. Cao, F. Verbeek, et al. Neural networks for increased accuracy of allergenic pollen monitoring. *Scientific Reports*, 11(1):11357, 2021. doi:10.1038/s41598-021-90433-x.

[15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL: http://arxiv.org/abs/1506.02640, arXiv:1506.02640.

[16] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee. Yolo26: Key architectural enhancements and performance benchmarking for real-time object detection, 2026. URL: https://arxiv.org/abs/2509.25164, arXiv:2509.25164.

[17] R. Sapkota and M. Karkee. Ultralytics yolo evolution: An overview of yolo26, yolo11, yolov8 and yolov5 object detectors for computer vision and pattern recognition, 2026. URL: https://arxiv.org/abs/2510.09653, arXiv:2510.09653.

[18] C. Stiekema. One-stage neural network analysis for allergenic tree pollen counting. Master's thesis, Leiden University, Leiden, The Netherlands, August 2025.

[19] Ultralytics. Hyperparameter tuning, 2025. Accessed: 2026-06-14. URL: https://docs.ultralytics.com/guides/hyperparameter-tuning/.

[20] Ultralytics. Customizing trainer, 2026. Accessed: 2026-06-24. URL: https://docs.ultralytics.com/guides/custom-trainer.

[21] Ultralytics. Transfer learning with frozen layers, 2026. Accessed: 2026-06-25. URL: https://docs.ultralytics.com/guides/custom-trainer#freezing-and-unfreezing-the-backbone.

[22] Ultralytics. Yolo26 vs yolov8 comparison, 2026. Accessed: 2026-06-24. URL: https://docs.ultralytics.com/compare/yolo26-vs-yolov8.

[23] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019. URL: https://arxiv.org/abs/1911.11929, arXiv:1911.11929.