UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Denys Kaliuzhnyi

Reducing the Effect of Incomplete Annotations in Object Detection for Histopathology

Master's Thesis (30 ECTS)

Supervisor(s): Mikhail Papkov, MSc
Dmytro Fishman, PhD

Tartu 2023

Reducing the Effect of Incomplete Annotations in Object Detection for Histopathology

Abstract:
Histopathology is a crucial component of clinical practice involving microscopic tissue examination. Typically, pathologists manually analyse tissue to locate and label structural units, cells, and organoids. The properties and quantity of these objects can indicate a patient's condition, e.g., the presence of tumours. Recent advancements in artificial intelligence (AI) have created the potential to automate this process. However, AI methods either provide limited accuracy or require a lot of densely annotated data, which is prohibitively time-consuming and expensive in the histopathology domain due to high object density and labelling difficulty.

In this study, we address the challenge of training object detection neural networks on histology data with incomplete annotations. We demonstrate that hyperparameter tuning can mitigate the negative effects of sparsely labelled data. Additionally, we propose a novel model component called the Generalised Background Recalibration Loss to further improve detection rates. It can be adapted to a broader class of object detection models than previous solutions.

Our results should facilitate the development of object detection neural networks for histology images by demonstrating the efficient use of sparsely labelled data. Our method reduces the impact of missing annotations on detection rates and thereby eases the most time-consuming aspect of data preparation for neural network training.

Keywords:
deep learning, computer vision, neural networks, object detection, sparsely annotated objects, training under incomplete annotations, histopathology, microscopy imaging

CERCS: P176 – Artificial intelligence; T111 – Imaging, image processing; B110 – Bioinformatics, medical informatics, biomathematics, biometrics

Mittetäielike Annotatsioonide Mõju Vähendamine Histopatoloogia Objektide Tuvastamisel

Lühikokkuvõte: Histopatoloogia on kliinilise praktika oluline komponent, mis hõlmab kudede mikroskoopilist uurimist. Tavaliselt analüüsivad patoloogid kudesid käsitsi, et leida ja märgistada struktuuriüksused, rakud ja organoidid. Nende objektide omadused ja kogus võivad viidata patsiendi seisundile, nt kasvajate olemasolule. Hiljutised edusammud tehisintellekti (AI) vallas on loonud potentsiaali selle protsessi automatiseerimiseks. AI-meetodid pakuvad aga kas piiratud täpsust või nõuavad palju tihedalt annoteeritud andmeid. Annoteeritud andmete vajadus on histopatoloogiliste objektidesuure tiheduse ja märgistamisraskuste tõttu ülemäära aeganõudev ja kulukas.

Selles uuringus käsitleme närvivõrkude koolitamise väljakutset mittetäielike annotatsioonidega histoloogiliste objektide andmete tuvastamiseks. Näitame, et hüperparameetrite häälestamine võib leevendada hõredalt märgistatud andmete negatiivseid mõjusid. Lisaks pakume välja uudse mudelikomponendi, Üldistatud Tausta Ümberkalibreerimist, et veelgi parandada tuvastamissagedust. Seda saab kohandada laiemale objektituvastusmudelite klassile kui varasemaid lahendusi.

Meie tulemused peaksid hõlpsustama närvivõrkude arendamist histoloogiliste piltide objektide tuvastamise jaoks, näidates hõredalt märgistatud andmete tõhusat kasutamist. Meie meetod vähendab puuduvate annotatsioonide mõju tuvastamismääradele ja lihtsustabseeläbi närvivõrkude koolitamise andmete ettevalmistamise aeganõudvaimat aspekti.

Võtmesõnad:
sügavõpe, tehisnägemine, neurovõrgud, objektituvastus, hõredalt annoteeritud objektid, koolitus mittetäielike annotatsioonide alusel, histopatoloogia, mikroskoopiakujutised

CERCS: P176 – Tehisintellekt; T111 – Pilditehnika; B110 – Bioinformaatika, meditsiiniinformaatika, biomatemaatika, biomeetrika

Contents

1 Introduction

Microscopy imaging is a crucial tool in modern medicine that opens up the possibility of treatment development, disease exploration, and investigation [7]. Many microscopy studies require manual labelling and classification of objects observed in a tissue sample. Such annotation is usually time-consuming, error-prone, and requires prior professional training. Automated microscopy image analysis can aid by speeding up medical specialists' workflow and improving diagnostic accuracy [38, 37]. Such automation is usually powered by large quantities of imaging data, where the expert annotates objects of interest. Images and labels combined create a dataset that serves as an input to algorithms that search for patterns in data and their relation to annotations. These algorithms produce so-called *trained models* that can run on new incoming images and output predictions based on extracted knowledge.

Software like Ilastik [55] and CellProfiler [59] provides a simple interface to automate a variety of histology image analysis tasks (e.g., cell detection, segmentation, classification). Such programs embed a list of common tasks (i.e., templates) and corresponding methodologies, the complexity of which is hidden from the user. In a common workflow scenario, the user iteratively adds annotations to image regions, bringing the most valuable gain to model performance until saturation. Alternatively, more advanced automation solutions can be based on ad-hoc neural network development. The latest discoveries in artificial intelligence [58, 36] have shown a huge potential to advance microscopy image analysis. Neural networks allow for building more custom models that typically work faster and more accurately than out-of-the-box solutions.

1.1 Problem

Template software for automated histology image analysis can usually provide only limited accuracy. The reason is that these tools incorporate decision models that cannot perform equally well on all possible input data. Sometimes, a more detailed selection or customisation of the model is required to achieve the best performance. In addition, embedded algorithms are frequently outdated compared to the current state-of-the-art solutions. Lastly, the possibilities of the software application are bounded by the built-in functionality, which sometimes cannot cover specific needs. On the other hand, these tools have the advantage of being able to achieve decent accuracy with low volumes of annotated data. Pathologists do not need to annotate every object of interest present in the image. As a result, they save time by only focusing on a critical portion of data, producing *sparse* annotations. Such simplification is essential for the histopathology domain, where object density in the image can be very high.

In contrast, custom solutions based on neural networks can adapt to a wide range of tasks and provide state-of-the-art output accuracy. The speed of some neural networks enables real-time computing on edge devices [18, 57, 20]. Moreover, they are flexible and

can adjust the trade-off between model speed and accuracy. However, their performance highly depends on the volume of data they are built on. Most importantly, these models are trained upon the assumption that the annotation data is *dense*, meaning that every relevant object is properly identified and labelled in the image. If this assumption does not hold, missing annotations send misleading signals to the model, which hurts performance.

In this work, we search for a solution that keeps the positive features of existing methodologies and eliminates their weaknesses. That way, we attempt to solve the problem of efficient neural network training on histology data in sparse annotation settings. Our central hypothesis is that modern neural networks can perform well on histology data even under extreme annotation incompleteness, especially when dedicated model adjustments are employed.

1.2 Motivation

Sparse annotations are common in histopathology due to the image's high labelling cost and rich object density. We aim to relax the requirement of dense (i.e., complete) annotations in object detection neural network training. Efficient model training on sparse annotations will significantly free specialists' time for data preparation while preserving the use of the latest artificial intelligence methods.

Although there were works that suggested special methodologies to train object detectors in sparse annotation settings [67, 42, 62], they primarily tested proposed solutions on natural image datasets like PASCAL VOC [4] and COCO [32], which are substantially different from histology images. Consequently, our work aims to test the performance of such methods in the histopathology domain and see if they bring the same or similar gain to the model detection rate.

1.3 Contribution

In this work, we study the performance of state-of-the-art object detection neural networks in sparse annotation settings for histopathology. We conduct a model hyperparameter tuning to aid in training. We also adapt techniques from previous works and propose a new solution to alleviate neural network training under extreme annotation incompleteness (up to 95% missing). In particular, we propose the Generalised Background Recalibration Loss neural network component that extends previous work [67] by making the implementation compatible with a broader range of models. As a result, we demonstrate that efficient object detection on histology data is feasible even under significant annotation incompleteness, especially with the employment of dedicated methodologies.

2 Writing assistance

Grammarly and ChatGPT assisted in the writing of this thesis. Grammarly, a cloud-based typing assistant, uses AI technology to check for grammar, spelling, punctuation, clarity, engagement, delivery mistakes, and plagiarism detection [2]. We utilised Grammarly to correct grammar mistakes, restructure sentences for better clarity, and properly punctuate the text. ChatGPT is a chatbot that utilises the Generative Pre-trained Transformer (GPT-3.5) language model and can perform tasks such as question answering, generating text, summarising articles, and translating text [3]. We employed ChatGPT to paraphrase sentences in a more academic manner and fix grammar mistakes.

3 Background

In this section, we first make an overview of the histopathology domain and present the Whole Slide Image data representation format used in our work. Then, we introduce deep neural networks and focus on solutions and peculiarities of object detection, including architecture and training aspects. Afterwards, we discuss YOLOv5 [22] state-of-the-art object detection model we based our experiments on. Lastly, we describe previous works that studied the methods of improving object detection performance in missing annotation settings.

3.1 Histopathology imaging

Histopathology is the study of changes in tissue caused by the disease. Histopathologists often employ microscopes to carefully study the tissue samples extracted from patients to make diagnostic or treatment decisions. Histopathology can focus on various parts of the human body. Organs like the skin, liver, kidney, breast, bladder, colon, and stomach are all viable subjects for histopathological examination if non-invasive methods do not provide enough information for diagnosis. Histopathology has a wide range of applications in clinical practice. Most commonly, it is used to provide a precise diagnosis. It can help to diagnose diseases like ulcerative colitis, Crohn's disease, cancer and infections. In all cases, tissue must be first extracted i.e., biopsied and then preprocessed before the pathologist can examine it.

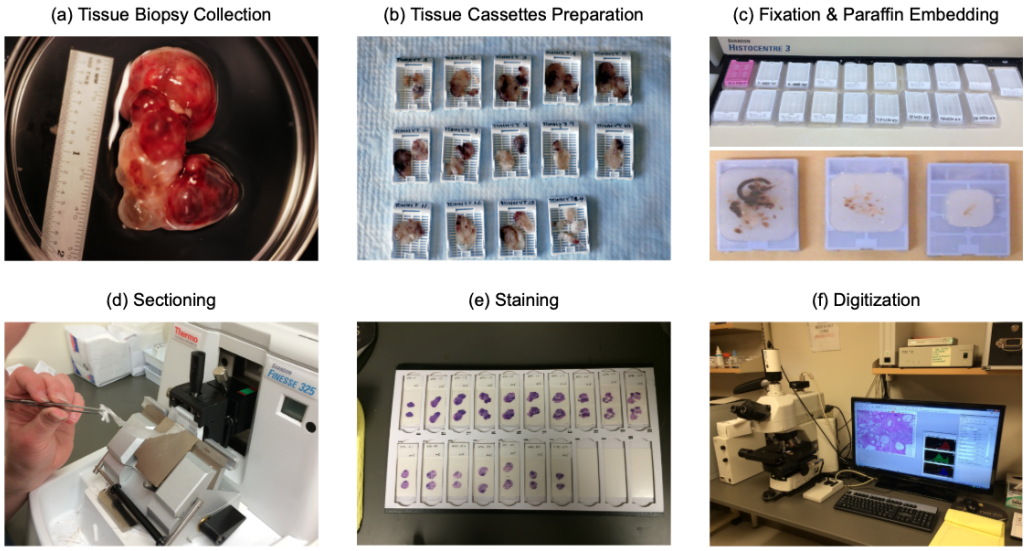

Figure 1. Stages involved in biopsy preparation [39].

A biopsy is followed by multi-step processing to make raw tissue suitable for examination. The exact methodology can vary depending on the laboratory setting. However, generally, the process comprises the following steps: cutting tissue samples into smaller pieces, dehydration, fixation in paraffin medium, sectioning (i.e., slicing into pieces of 3-4 µm thickness), and staining to enhance the visibility of desired objects. Digital pathology also includes a digitization step at the end. All stages are illustrated in Figure 1.

3.1.1 Digital pathology

Digital imaging is a method of creating a digital representation of the visual characteristics of real-world objects. Just like a regular smartphone takes the scenic view of the sunset, there are devices designed to electronically capture the appearance of tissue. Digital pathology is the study of digital representations (i.e., images) of tissue samples that uses dedicated hardware and software, usually a microscope with an embedded camera. That way, image of obtained biopsy can be preserved on a computer for later examination. The benefits of such an approach are evident: there is no need to keep the original tissue as the corresponding digital copy is always available. Also, one can use specialised algorithms to automatically analyse it.

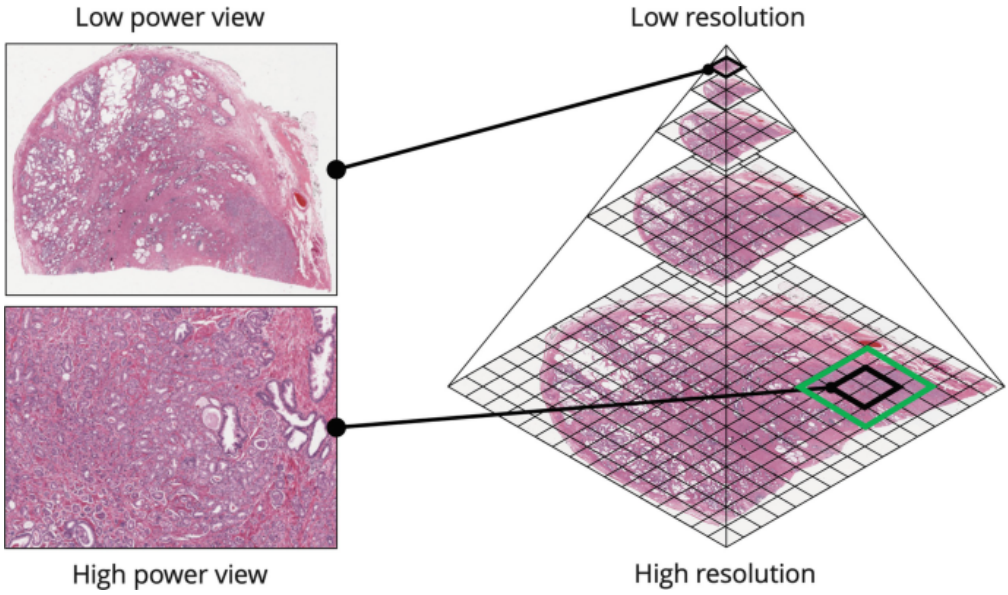

Figure 2. Whole Slide Image format structure [40].

So-called virtual slides gained a lot of popularity across various applications of pathology. Whole Slide Image (WSI) is a widespread format for storing digital images of tissue samples. It was introduced in 1999 and became a handy and efficient way of

storing high-resolution views of the entire tissue on a glass slide. The critical feature of the WSI format is its hierarchical structure. During tissue capturing, the microscope's software takes images of the tissue slide at multiple resolution levels. In other words, several independent images at varying magnifications describe biopsy with different levels of detail. These magnifications are usually referred to as powers because each subsequent zoom level gains a double increase in resolution. WSI format structure is illustrated in Figure 2.

WSI format may seem overly complex and has a number of downsides. First, storing only the image corresponding to the largest magnification scale would be sufficient to comprehensively explore the obtained tissue sample. Hence, removing unnecessary representations would reduce the file size overall. Secondly, such a complex file format makes it hard to operate. WSI requires dedicated software to open and examine the content. However, the hierarchical structure is necessary to lift computational constraints, because the largest magnification level frequently occupies a lot of disk memory (up to hundreds of gigabytes). Hence, loading the whole image would be unfeasible on most workstations. The magnification pyramid of the WSI makes it possible to dynamically adjust zoom levels when a pathologist examines the slide. Fine-grained representations are loaded into memory when a user zooms in on the image, and smaller powers are loaded for a zoomed-out view. The organisation of this process is analogous to view scaling in digital world map applications.

3.1.2 Image processing

Image processing algorithms help to prepare the input data for downstream analysis, including training the AI models. In histopathology, image processing is more complex than in many applications due to complicated WSI data format. Firstly, The entire WSI slide is rarely used as input to the AI model. Often, information is taken from a particular WSI magnification level. For example, minor structure analysis like individual cells is typically conducted on higher resolution levels as $\mathrm{x}20$ or $\mathrm{x}40$. Lower resolutions can be helpful when performing high-level investigation. Secondly, the fine-grained resolutions are too large for a direct AI model input, reaching hundreds of thousands of pixels in both dimensions. Even modern computational hardware cannot handle so large inputs because of memory and computational restrictions. Typical input size for modern AI models is hundreds or, at most, thousands of pixels. Hence, WSI files need to be preprocessed for downstream analysis.

The key processing step for WSI data is patch generation. The patch, which is also referred to as an image crop, is a sub-region of the initial large-resolution image. The size of this sub-region is fixed by the human engineer. For example, common resolutions are 224×224 and 512×512 pixels. Generally, AI models trained on larger-size patches perform better because of broader context integration. Patch sampling methodology depends on the exact application, but generally, it is either random or integrates information

about annotation locations to only select labelled regions. Sampling can be done with overlaps (meaning that different patches can share some number of pixels) or without, e.g., splitting an initial image into a tile-like grid of non-overlapping patches. Mentioned above methodologies are shown in Figure 3.

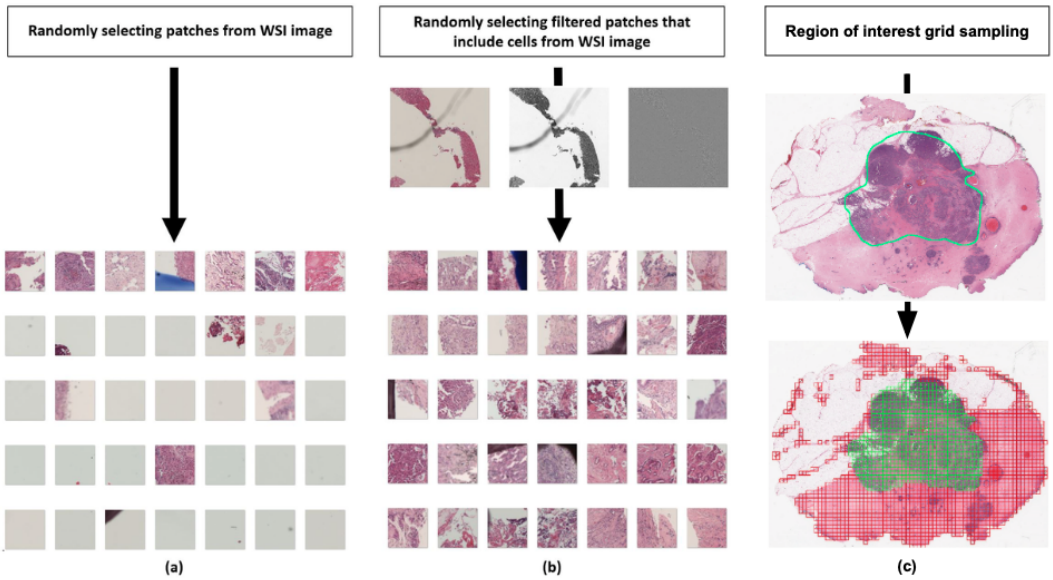

Figure 3. Methodologies for generating patches based on high-resolution WSIs [11, 25].

3.2 Neural networks in deep learning

Machine learning is a computer science subfield that takes advantage of dedicated algorithms to let computers learn from the provided data. It encompasses many methods that allow one to solve a wide range of tasks. These methods are classified into the following four groups: supervised learning (each input data sample X is accompanied by a know target value y we aim to predict, so-called labelled data), unsupervised learning (only input X is provided, so-called unlabelled data), semi-supervised learning (a hybrid of two previous: combination of labelled and unlabelled data), and reinforcement learning (there is no data prepared in advance; instead, the so-called agent interacts with the predefined environment to obtain X and y samples). In supervised machine learning, which is in the scope of our work, the most common tasks are classification and regression. Classification aims to assign a class label to the entity where a set of labels does not possess any relative ordering. In contrast, regression assigns a continuous label to the entity, which provides a scale to rank them.

Deep learning is a machine learning subfield that employs artificial neural networks as learning algorithms. Neural network can be defined as a function $f : \mathbb{R}^n \to \mathbb{R}^m$,

which is a transformation of input $X \in \mathbb{R}^n$ to an output $Y \in \mathbb{R}^m$. From an architectural point of view, a neural network is a sequence of layers where input data is transformed and passed within them to produce a desired output. In this sequence, the very first layer is called the input layer, and the last one is the output layer. All the others in between are hidden layers. The number of layers is usually referred to as the depth of a neural network. Deeper neural networks can learn more complex features from data but are also prone to overfitting (memorizing training data) and have other problems like vanishing gradients [17]. The particular connection of layers defines the information flow route. Some neural network architectures use a strict layer connection sequence order where information is passed from one layer to a succeeding one and no others [27]. Other architectures allow more sophisticated information flow that can be described with directed acyclic graph [30, 50]. In more specific application cases, circuit types of connections can take place.

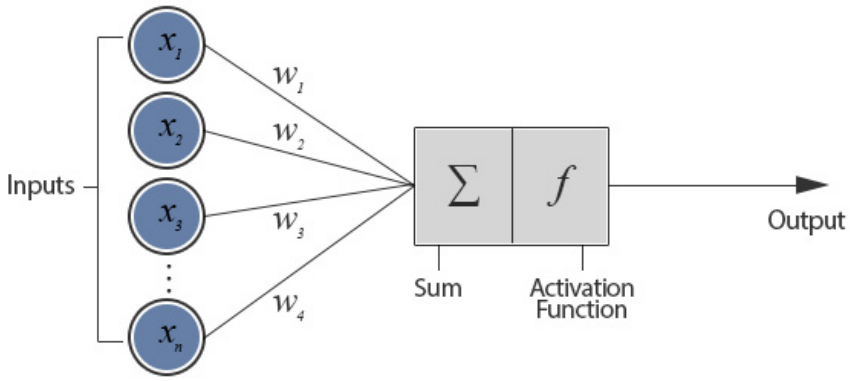

Figure 4. Activation of the output neuron. Input neuron activations x_i are multiplied with corresponding synapse weights w_i, then summed up and passed through the activation function f [19].

Each layer inside contains a number of structural units — neurons. A neuron is responsible for transmitting particles of information throughout a network. The transmission is conducted via synapses — connections between neurons that hold learnable parameters. When information flow reaches a particular neuron, we call this event a neuron activation. In a typical scenario, neuron activation is calculated via summing up a pairwise multiplication of connected inputs and associated synapse weights and then applying an activation function, which usually introduces non-linearity (see Figure 4). In a feed-forward network, synapse weight connects neurons between layers, but it can be used within layers in different architectures like recurrent neural networks. In addition, layer connection density can vary much. For instance, in a fully connected neural network (FCN), each neuron of one layer is connected to all the neurons of the

subsequent layer. In contrast, synapses between the convolutional layers are sparse (not fully-connected) and shared between multiple input and output neurons. Figure 5 shows a simplified example of FCN architecture.

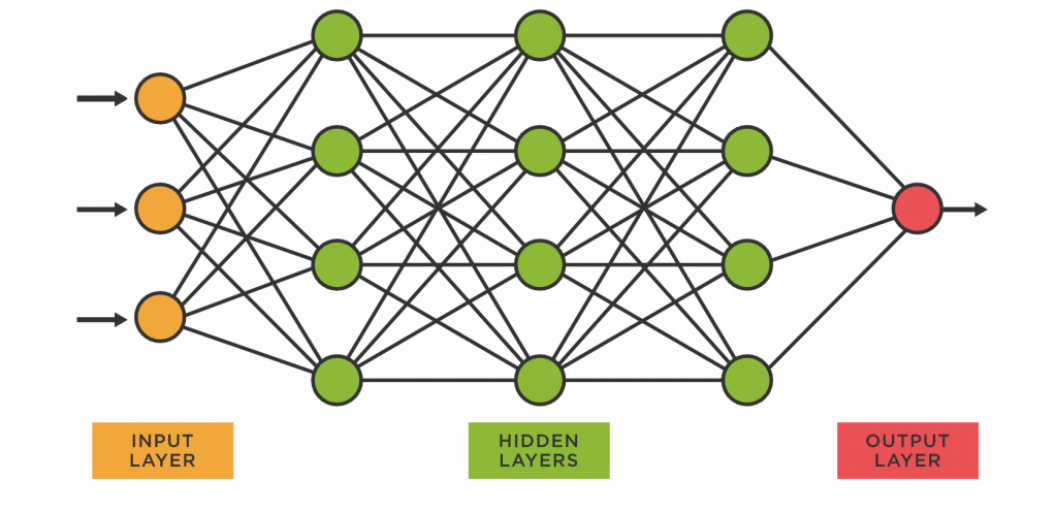

Figure 5. Example fully-connected network with three hidden layers of four neurons each. The input layer accepts three units of information, and the output layer returns a single value [6].

Neural networks would only be helpful with an efficient algorithm to train them. The loss function is an integral part of the training procedure. This function serves as a measure of the match between the network's real and desired outputs. The point of neural network training is to minimise the loss value as much as possible. The choice of the loss function is tightly bounded to the type of task the network solves. For example, the most common loss function in classification is cross-entropy (i.e., negative log-likelihood), while in regression, it is a mean squared error. Differentiability is the key demanded property of the loss function that makes it suitable for training.

The conventional network training procedure is comprised of two main steps: forward pass and backward pass. The forward pass (also called inference or prediction) is simply a transition of the data throughout the neural network from input to output layers. In an ideal setup, we expect a model to give us a correct output (e.g., classify an image as the one showing a dog). However, that most probably will not be the case if a model is not trained yet. At this point, the second stage of actual learning begins. The purpose of the backward pass is to utilise loss function to adjust the model parameters (including connection weights but not limited to them) in order to make the prediction match the desired value. Backward pass is associated with backpropagation — a crucial and efficient algorithm for neural network training.

Backpropagation [51] begins by applying the loss function: predicted and expected values are used to calculate the loss representing the error measure. Further, this error is propagated throughout the network layers in a reversed direction using the chain rule of differentiation. The backpropagation aims to find the weights responsible for deviating the output value from the desired one and then assigns adjustment coefficients that should move the model parameters toward producing the aimed output. The network's weights adjustment itself is accomplished with the help of the optimiser. There are many, but stochastic gradient descent (SGD) and Adam [26] are two common examples. Optimiser moves the parameter values in a direction opposite to the gradient with a step size proportional to the magnitude of the gradient itself and an adjustable learning rate parameter. Learning rate can be either a constant or defined by a so-called scheduler function, e.g., One Cycle [54] or Cosine [35] schedulers.

3.3 Convolutional neural networks in computer vision

Computer vision is a subfield of computer science that deals with information extraction from image data. For instance, computer vision can be used to perform edge detection for fingerprint matching or detection of pedestrians crossing the road. The complexity of the tasks can vary much, as well as the applied methods. In general, neural networks are used in tasks that require complex pattern recognition, semantic understanding of the scene, knowledge integration, etc. For example, instance segmentation of cars on the road or generating new images from a text prompt require the application of sophisticated neural networks. Otherwise, more simple tasks can be solved using dedicated algorithms. For example, Hough transformation algorithm [12] works well for line and circle shape detection.

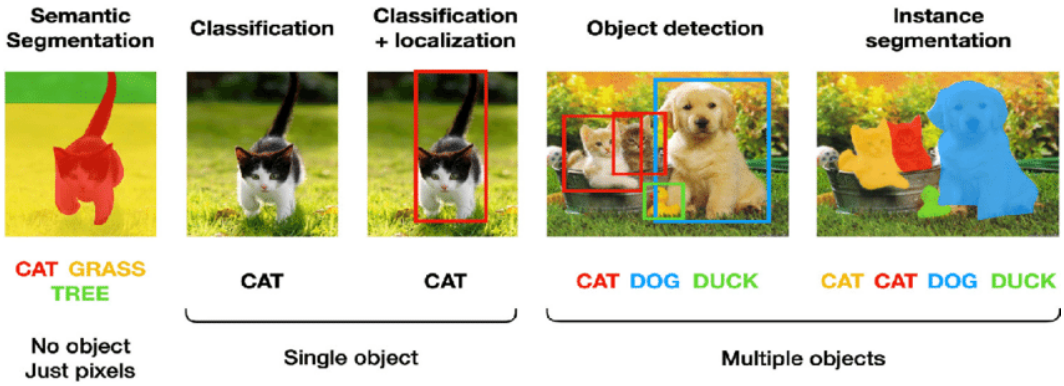

Figure 6. Widespread tasks in computer vision [23].

The most popular tasks in computer vision are image classification, object detection, and image segmentation (see Figure 6). Image classification is a particular case of

classification (described in Subsection 3.2), where input is image data. In addition, the task can also include localisation (drawing the bounding box) of the object that reaffirms the class. Object detection is similar to classification with localisation but assumes localising multiple objects of possibly varying classes in a single image. In other words, object detection combines regression on parameters of bounding boxes and classification of their labels. Image segmentation goes further with the granularity of the image analysis. It assigns class labels on a pixel level, which results in a meaningful mask prediction that can be overlaid with the original input image. Image segmentation can be semantic and instance-based. The former does not segregate pixels based on the objects they belong to (e.g., pixels can be of the same class *cat* but belong to different cats). In contrast, instance segmentation identifies individual objects.

A convolutional neural network is a particular type of network designed to deal with spatially correlated data. It is based on convolution, a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other. In the case of CNN input, one function represents the so-called kernel (or filter), and the other is the input image. The kernel is defined as a matrix, usually way smaller than the image it is applied over. The kernel is iteratively shifted across spatial image dimensions with step size defined by stride parameter: from left to right and from top to bottom. For a particular image pixel location, convolution is calculated as a dot product of a kernel rotated by 180 degrees and the corresponding image region that overlaps with the kernel. An example of a convolution is illustrated in Figure 7. The kernel in convolution operation acts as an extractor of the particular type of information. CNNs utilise multiple kernels in order to let each of them focus on different information patterns.

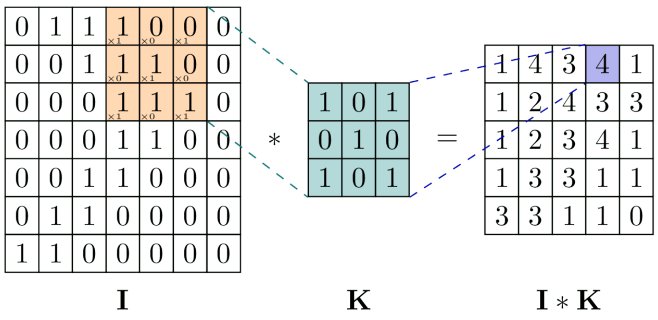

Figure 7. Convolution operation on image I with filter (kernel) K [49].

One reasonable interpretation of convolution in CNNs is that a layer structure of a network turns convolution into an iterative information merge procedure. Previous experiments showed [66] that CNNs tend to extract simple features (e.g., edges, basic shapes like circles or rectangles) from an image in the early layers (closer to the input) and more complex morphologies (e.g., faces) in the end layers. The hierarchical nature of the objects justifies such behaviour: faces are comprised of simple line and circle shapes, and the face itself is part of another object — a person. Besides convolutions, there are also different types of layers that can be crucial for many CNN architectures. For

example, these are pooling layers (rapidly reducing spatial dimensionality of the image to speed up training), normalization layers (re-scales distribution of layer activations to make training more stable and fast), etc. In addition, fully connected layers are often applied on the top of a CNN for some computer vision tasks like image classification.

The main difference between CNN architecture compared to FCN is that convolution's connections are not dense. The same synapse weights are used to extract spatial information from different parts of the image. That approach highly reduces the training time and model size. In addition, pattern recognition becomes position-independent in CNN. That means if a network learns to detect a dog in the centre of the image, it will most probably be able to find it in all the other possible locations.

Figure 8. VGG-16 CNN architecture. Each block describes kernel size (except fully connected layers), operation (convolution or max-pooling or fully connected layer), activation function (for convolution only), and output feature map resolution. [24].

A VGG-16 CNN architecture [53] can be seen in Figure 8. That is an example of a tiny and straightforward structure model for illustration purposes. A VGG-16 model consists of convolution, max-pooling and fully connected layers. Layer connections are strictly sequential, and model depth accounts for 16 layers, as the model name stands for. Note that pooling layers are not counted for network depth because they do not own trainable parameters.

To illustrate the input image transformation evolution throughout the network, Figure 9 shows how the input is being modified from layer to layer of the VGG-16 neural network. Initially, the input is a matrix of size 224×224 representing a grayscale image (channel dimension equals one). Then, succeeding layers tend to squeeze spatial image resolution and enrich channel dimension. In the end, input data is turned into 1000-dimensional output vector. The such architectural pattern of image transformation is common to many CNNs. Although, some models have no fully connected layers. For example, an image segmentation model instead frequently performs a reversed operation of upsampling of reduced-size images to restore the input spatial resolution.

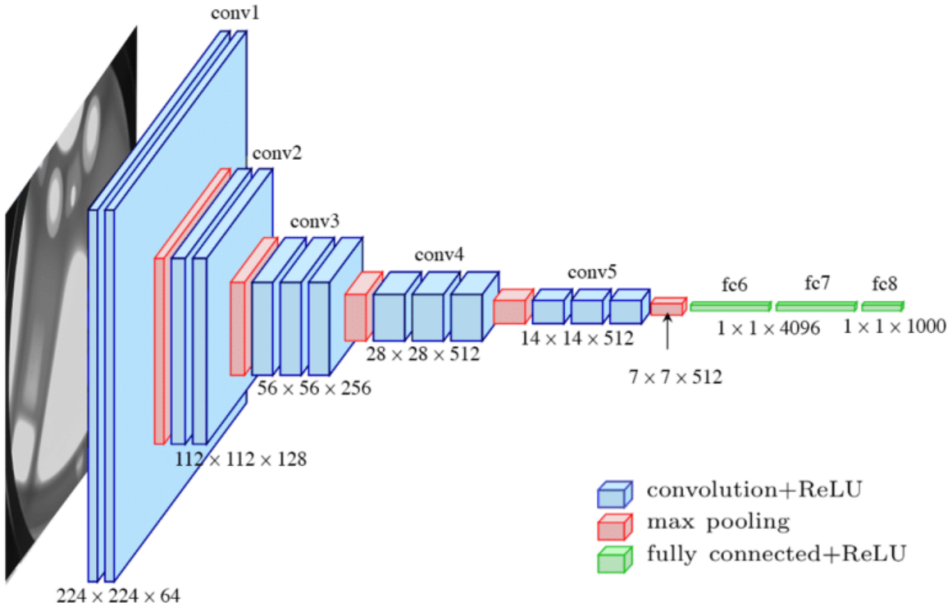

Figure 9. VGG-16 image transformation. Text subscriptions at the top denote layer names, and at the bottom are transformed data resolution of the indicated layer [13].

Nowadays, state-of-the-art CNNs are often very deep, with hundreds of layers. Networks often incorporate complex structure units (often referred to as neural network blocks, e.g., inception blocks [64]) with non-trivial connections between them (e.g., spatial pyramid pooling layers [16]). Architectural patterns can vary much and usually depend on computer vision task network solves. But they all share the idea of efficient feature extraction through convolution operation and utilise knowledge of hierarchical feature build.

3.4 Object detection

As already described in Subsection 3.3, object detection is a widespread task in computer vision that deals with identifying objects in the image. The detection usually assumes drawing a bounding box around the object of interest and then classifying it in the case of necessity. The application of object detection is vital in many fields, like autonomous driving, robotics, medicine, etc.

There are many approaches to detecting objects in the image, but nowadays, most successful methods are based on deep neural networks. Detection CNNs are also classified depending on the architectural patterns. There are one-stage and multi-stage (e.g., two-stage) detectors. The former ones are known to be faster and easier to train compared to the latter ones [10, 34]. However, two-stage detectors usually provide

higher detection accuracy [10, 34]. This work is focused on the application of one-stage detectors.

This section first introduces the standard structure of one-stage detectors. Secondly, it describes the most common evaluation metrics for object detection tasks, which we used in our experiments. Thirdly, we explain peculiarities and give examples of the loss functions utilised in object detector training. Fourthly, we describe the approach of the You Only Look Once (YOLO) method [44] and briefly describe the members of the YOLO model family. Finally, we present the YOLOv5 model [22] used in our experiments in more detail.

3.4.1 One-stage detectors

One-stage detectors perform inference in a single-stage fashion. In the opposite case, an intermediate component is responsible for generating region proposals, which will be further refined and classified by the main network. In contrast, one-stage detectors employ a single end-to-end seamless image transformation sequence to extract predictions. Such model architecture provides more straightforward and faster model training. Fast test time prediction of one-stage models enables real-time object detection on edge devices.

One-stage detectors share many standard architectural features. First of all, they are usually comprised of the following structural components: backbone, neck and head. Secondly, many detectors take advantage of non-max suppression algorithm. Additionally, anchor boxes are widely used. We will explain each concept in detail.

Detector backbone. A backbone in object detection is a feature extraction network (typically CNN) that serves the crucial function of feature representation learning. Popular architectures developed for image classification often find application in object detector backbones. For example, ResNet [17], VGG [53], and MobileNet [18] model families are frequently used. Alternatively, there are CNN models designed specifically to work best in object detection tasks, for example, DarkNet19 model [45], a variant of the ResNet architecture. In both scenarios, pretraining on a classification task for these models and then utilizing obtained weights as a starting point for training detection backbones is common.

Detector head. The detector head is also an integral unit for every CNN. It is a subnetwork that resides at the very end of the model and contains output layers. Detection head build varies depending on the particular model type. Nevertheless, in all cases, it outputs bounding boxes and object class scores. Bounding boxes are usually represented as centre or corner coordinates plus spatial dimensions, accounting for four parameters. Object class score can be a conditioned confidence score C, but some architectures also provide object presence probability score in a separate field, also

known as objectness score. In such case, object confidence score C is modelled as $C = P(obj, cls) = P(cls|obj)P(obj)$, where $P(cls|obj)$ is a predicted class probability (under the assumption that object exists) and $P(obj)$ is the probability of object presence regardless the class. Lastly, $P(obj, cls)$ is the probability of the object presence of the particular class, which is treated as prediction confidence C.

Detector neck. The neck is an optional intermediate structure between the detector backbone and the head. It is usually implemented as an additional set of layers dedicated to collecting and integrating feature maps from different backbone layers. Usually, so-called top-down and bottom-up paths in the neck combine feature representations from different abstraction levels. For example, Feature Pyramid Network (FPN) [30] is a popular choice for the detector neck. Modern architectures frequently utilise it to substantially improve the detection rate [46, 9, 20, 15, 47].

Non-max suppression. A non-max suppression (NMS) algorithm is crucial for many architectures. It is applied over the predicted bounding boxes as the last step. NMS iteratively eliminates box candidates by considering highly overlapped pairs. An intersection over Union (IoU, see details in Subsection 3.4.2) threshold parameter defines a sufficient overlap to consider candidates. In each pair, the box with lower predicted confidence is filtered out. Then the process repeats until no pairs above the threshold remain. The application of NMS for some models is necessary because detectors often tend to produce an abundance of boxes.

Anchor boxes. Anchor boxes are a set of predefined boxes of a certain width and height. In some detector architectures, output bounding box dimensions are regressed directly. However, this approach often performs poorly when the dataset contains objects of irregular shapes and scales. Anchor boxes aim to let each anchor specialise in different object sizes and aspect ratios. That way, we replace direct box dimensions regression with regression on anchor boxes. Hence, the model learns to predict offsets for particular anchor box instances. That solution also enables the increase of possible object density in the image. In case two objects highly overlap, especially when two centres are nearby, many models struggle to detect both due to architectural limitations. However, with anchor boxes, two objects can now be located very close to each other in the image. The model will easily handle such cases because separate anchors will be responsible for detecting two boxes.

3.4.2 Evaluation metrics

Model evaluation metrics are a necessary utility that enables model training tracking and summarizing obtained performance. They are also important because they are used for comparing and ranking different models. Metrics are usually designed to be intuitive and complete, which means a comprehensive reflection of model performance. Although, the standard metrics used in object detection are far from being simple and easy to capture at first glance. However, they are pretty robust as for the methods that estimate such complex systems of object detection with a single number.

A conventional metric in object detection is a mean average precision (mAP). This term name should not be perceived literally, meaning that metric is not about averaging precisions. We will explore the hidden implementation complexity step-by-step.

$$\text{Precision} = \frac{TP}{TP + FP}; \quad \text{Recall} = \frac{TP}{TP + FN} \tag{1}$$

In order to estimate model performance, we classify predicted bounding boxes into three cases. The first case is true positive (TP), which denotes objects detected and classified correctly. Then there are two types of errors: false positives (FP) and false negatives (FN). Detection is FP if the predicted bounding box has no corresponding ground truth object, meaning the prediction is wrong. Conversely, FN means that there is a ground truth object, but the model did not output the corresponding bounding box, meaning that the prediction is missing.

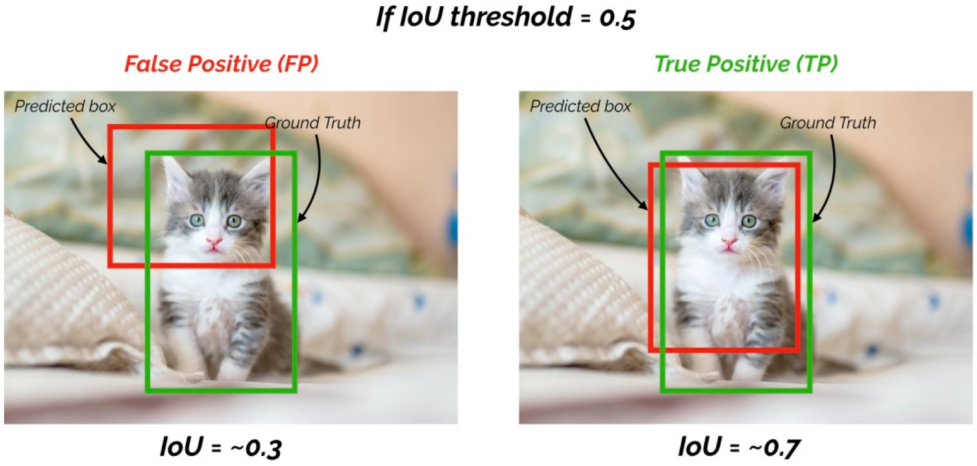

Figure 10. Example of Intersection over Union (IOU) application in object detection [65].

When the detection is done on a labelled image, the TP, FP and FN counts are summarised for each object class separately. Then, we derive the following two metrics:

precision and recall. Precision denotes the correctly identified object count ratio to the whole prediction count. Conversely, recall denotes the ratio of detected object count to the whole volume of ground truth annotations. The exact formulas are in Equation 1.

Returning to the prediction type identification, clarifying the logic behind box matching is essential. Calculation uses the Intersection over Union (IoU) metric to determine pairs of ground truth and predicted boxes that match. If IoU between boxes is larger than the predefined threshold, then the prediction is considered to correspond to the ground truth object. Otherwise, the prediction is either wrong or missing. The example is given in Figure 10. We conduct the matching for each object class separately so that we do not consider overlapped box pairs when ground truth and prediction belong to different classes.

Intersection over Union itself is a standalone metric used to estimate the overlap between two objects. IoU equals the intersection area between the objects divided by the area of their union. The formula is illustrated in Figure 11. That way, IoU equals 1 if and only if predicted and ground truth boxes ideally match. Conversely, IoU is 0 if and only if no intersection exists between objects.

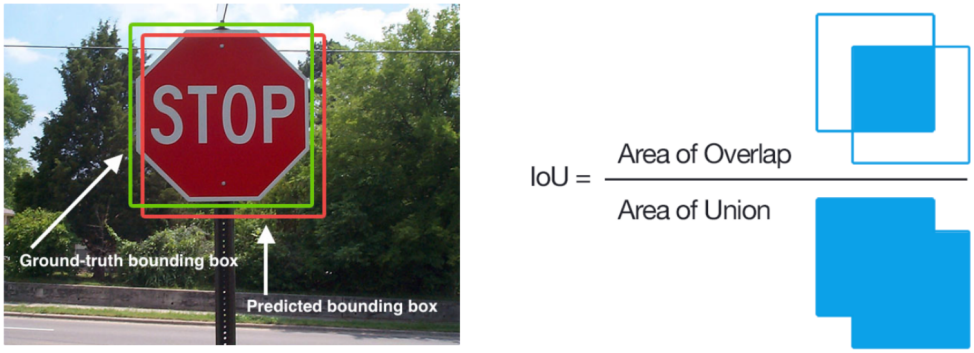

Figure 11. Intersection over Union (IoU) calculation formula [52].

The next step of mAP calculation is building precision-recall curves per class. We previously mentioned that number of TP, FP and FN is counted for each class prediction over the image. However, these counts can vary greatly if we change the model confidence threshold. This value is used to divide continuous model probability prediction into *yes* and *no* answers. Increasing the confidence threshold will result in higher precision but lower recall and vice-versa. The precision-recall curve is built by plotting these values in the 2-D plane obtained at different tabulated threshold values from 0 to 1. The purpose of the curve is to calculate the area under it, which will correspond to the average precision (AP) for the particular class. Larger AP means better model performance. The example curve is in Figure 12.

At this point, we obtained AP values per each class category. Lastly, the AP values are averaged out to obtain the final mAP score. Note that we initially assumed a particular IoU threshold to be used for accounting correct and wrong or missing predictions. Usually, this threshold equals 0.5. In order to explicitly showcase the threshold applied in calculations, it is often appended to the mAP acronym as follows: mAP_{50}. In addition, $mAP_{50:95:5}$ denotes the average of mAP scores obtained at a series of IoU thresholds: from 50% to 95% with step size 5%. It is considered to be more robust as accounts mAP scores at higher IoU thresholds. In other words, mAP_{50} is generally used to compare models from a detection rate point of view. On the other hand, $mAP_{50:95:5}$ score rather corresponds to bounding box localisation accuracy comparison.

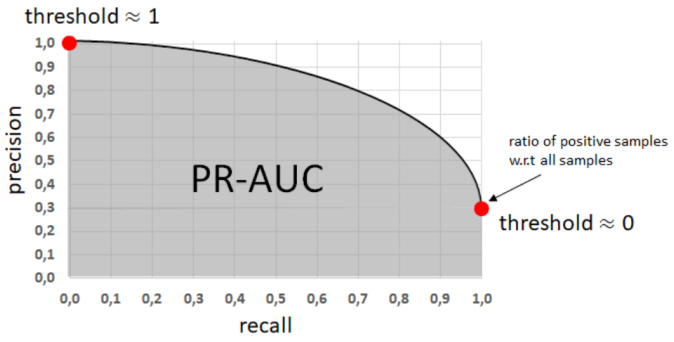

Figure 12. Typical appearance of the area under the precision-recall curve (PR-AUC) [5].

There is one important note regarding frequent confusion about the mAP metric. Terms mAP and AP are often used interchangeably, meaning that the word *mean* is optional in the metric name. However, to be more transparent, we denote AP as the score for the particular class category and mAP as their average. In the case of detection without box classification, there is no difference between these two terms.

Described mAP metric helps to estimate model performance. However, it does not provide any guidance regarding the optimal prediction confidence value threshold to use in testing. There is a different metric called F_1 score that aids the situation. It equals the harmonic mean of precision and recall. Its primary purpose in object detection is to select the model confidence threshold that will give an optimal trade-off between recall and precision. In a typical scenario, F_1 scores are calculated over the tabulated range of confidence thresholds, and then the one giving the highest F_1 is selected for the model detection testing.

3.4.3 Training losses

Training of object detectors is similar to other neural networks (described in Subsection 3.2). Although the application of the backpropagation algorithm and optimiser remains the same, the embodiment of loss penalty changes significantly compared to other tasks. Some detection models utilise specific types of loss functions to learn efficiently.

Many detectors use standard functions such as cross-entropy, mean squared error, and mean absolute error. Some solutions utilise a specific detection IoU box loss. The final

detector penalty is usually a compound of multiple losses calculated on separate output pieces. In particular, bounding box loss λ_{box} and classification loss λ_{cls} typically come separately. Moreover, if a model predicts objectness score apart from the classification one, then the objectness loss factor λ_{obj} is added as an additional component. Then, the total loss is calculated as $\lambda_{total} = a\lambda_{box} + b\lambda_{cls} + c\lambda_{obj}$, where a, b and c are calibrating weights. Such an approach is necessary to learn each output parameter of object detection.

Classification loss. Classification loss is essential for any object detector that classifies objects into categories. Usually, if the dataset is comprised of N object classes, then the network output contains N fields per predicted box to assign confidence for each class. The most common loss function used for classification prediction is a cross-entropy, also known as log-loss.

Objectness loss. If object presence probability is modelled separately from classification, the respective score should have an additional single output field. Like in classification, the conventional loss function for objectness score prediction is a binary cross-entropy (BCE) loss because objectness score prediction is a binary classification task. The formula of BCE is given in Equation 2, where y and p are ground truth and predicted values, respectively, ranging from 0 to 1.

$$\text{BCE}(y, p) = -y \log p - (1 - y) \log (1 - p) \tag{2}$$

Another common loss for the objectness score penalty is a focal loss (FL). It modifies BCE by adding the term that down-weights easy examples and lets the model focus on hard negative (i.e., confidently misclassified) cases. The formula of binary focal loss (BFL) is given in Equation 3, where γ is a hyperparameter to tune (higher values lead to more concentration on hard negatives). The standard default value for γ is 2.

$$\text{BFL}(y, p) = -y(1 - p)^{\gamma} \log p - (1 - y)p^{\gamma} \log (1 - p) \tag{3}$$

Box loss. The selection of the box loss is a more creative task. First, the favoured choice depends on whether anchor boxes are used. But in any scenario, bounding box coordinates can be learned in many ways. One option is common regression losses, for instance, variants of mean squared error or mean absolute error. Another option is classification losses, like cross-entropy, which is often employed in anchor-based architectures. It is important to emphasise that mentioned approaches directly regress box parameter values (x coordinate, y coordinate, box width, and height). That means these four parameters are treated independently from a loss point of view.

A different box loss is designed for object detection. IoU loss is derived from a differentiable implementation of the IoU metric. It is computed as one minus the mean of the

IoU scores between prediction and corresponding ground truth boxes (see Equation 4). IoU loss application is very intuitive. Instead of penalising predicted box parameter values directly, we integrally teach the model to achieve the initial goal of the object detector — to maximise the overlap between predicted and expected bounding boxes

$$\lambda_{IoU} = 1 - \frac{1}{N}\sum_{i=1}^{N} IoU(Pred_{box_i}, GT_{box_i}) \tag{4}$$

It is worth noting that, in practice, a pure IoU metric is unsuitable for neural network training. The reason is that if there is no overlap between the prediction and target, the IoU is 0, and hence there is no gradient to learn from. To aid this issue, the new variants of the IoU metric were developed to be used in IoU loss. For example, dedicated well-performing losses are Generalised IoU (GIoU) [48], Distance IoU (DIoU) [69] and Complete IoU (CIoU) [69]. They add extra components to classic IoU loss that enable non-zero gradients for non-overlapped predictions. For example, DIoU loss adds Euclidean distance factor between the centres of two boxes.

3.4.4 YOLO models

The most inherent example of one-stage detectors is the You Only Look Once (YOLO) model. The first YOLO paper was published by Joseph Redmon et al [44] in 2015. The main idea lies in performing detector training and prediction in one end-to-end stage. The model consists of the feature extraction backbone network based on GoogLeNet [56] and a fully connected layer head attached on top of it. The GoogLeNet is a typical CNN architecture formed from interleaving convolutional and pooling layers. Lastly, the NMS algorithm filters redundant boxes.

One of the leading architectural novelties of YOLO resides in its head component. The last output layer is designed to represent a grid of cells that can be logically mapped back to the input image regions. Mapping is done by dividing the original image into the $S \times S$ grid of evenly sized regions. This size corresponds to the output layer's spatial resolution. Then, the centre coordinates of an input bounding box determine the grid cell responsible for detecting the object. The whole process is displayed in Figure 13.

It is important to note that the output cell grid has a depth resolution in addition to a spatial one. That is essential because each cell must contain objectness score, box location and dimensions, resulting in a 5-parameter vector (c, x, y, w, h). Additionally, the architecture allows the prediction of multiple B boxes per grid cell to diminish the limited object density issue partially. Lastly, we need the score fields for each of the C classes. The complete shape of the output is defined as $S \times S \times (5 * B + C)$ tensor. In the original YOLO paper, the authors used $S = 7$, $B = 2$, and $C = 20$ because the model was trained on the PASCAL VOC [4] dataset with the corresponding number of classes. The model adopted mean squared error loss for learning the whole output tensor.

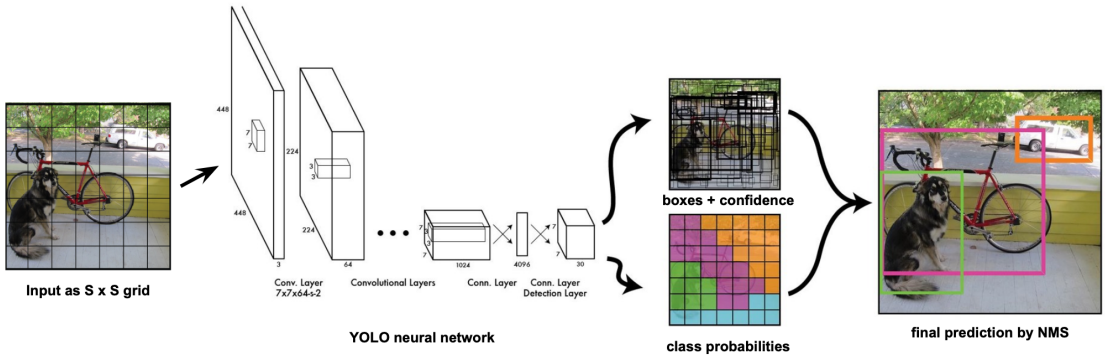

Figure 13. YOLO model architecture [44].

The subsequent development of the YOLO model was YOLOv2 [45]. It introduced many novelties that improved model speed and accuracy even further. Firstly, YOLOv2 replaces the backbone with the new Darknet-19 [45] architecture explicitly developed for object detection. Secondly, the model eliminates fully connected layers and integrates anchor boxes. Moreover, the incorporated algorithm automatically determines the best suitable anchors using dimensionality clustering over training data. Thirdly, the new model increases input image and output prediction resolutions. Lastly, YOLOv2 incorporates many other improvements. For example, each of the individual B boxes in a grid cell got a separate vector of C class scores instead of sharing a single one.

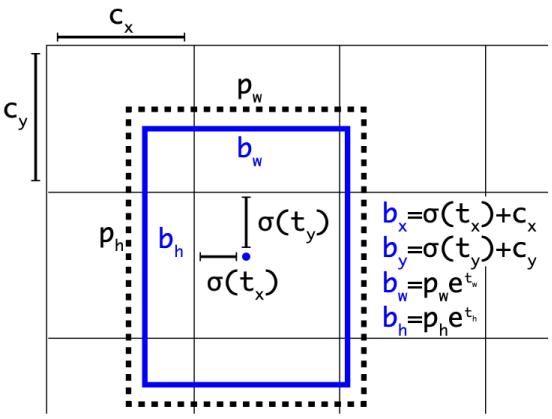

Figure 14. Anchor box to native space prediction transformation in YOLOv2 [45].

YOLOv2 anchor boxes are explained in Figure 14. The model predicts offsets t_x and t_y that represent the shift within an image grid cell. It also regresses t_w and t_h, which correspond to anchor box width and height offsets. Formulas in Figure 14 show how model drives native box coordinates (b_x, b_y) and dimensions (b_w, b_h). In the example, p_w and p_h denote anchor box width and height. Lastly, c_x and c_y are the spatial sizes of a grid cell.

The next generation of YOLO was YOLOv3 [46] which was an incremental improvement of YOLOv2. It updated the backbone architecture to Darknet53. Then, there are changes in loss functions: objectness and class scores are now penalised by a log loss instead of a mean squared error. But most importantly, YOLOv3 manifested the

usage of Feature Pyramid Networks as a model neck in the YOLO family. FPN makes efficient feature integration from different representation layers. Consequently, FPN allows YOLOv3 to make predictions at three different scales, meaning that multiple output tensors predict objects at different granularity levels.

The YOLOv4 [9] further refined the object detection accuracy and speed by introducing a broad list of so-called *universal features*. They include Weighted Residual Connections (WRC), Cross Stage Partial connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial Training (SAT), and Mish activation.

In summary, YOLO family models showed to be very efficient in terms of both speed and accuracy. The release of the prior version initialised a series of derived and improved YOLO models that have been developing since 2015, and new versions are still under research nowadays. Today YOLO model family accounts for dozens of instances.

3.4.5 YOLOv5

Since 2015 the Ultralytics team have been developing a new YOLO model based on the preceding YOLO architectures [44, 45, 46, 9]. As a result, the initial release of the YOLOv5 model [20] published in 2020 incorporated the best practices of ancestors and

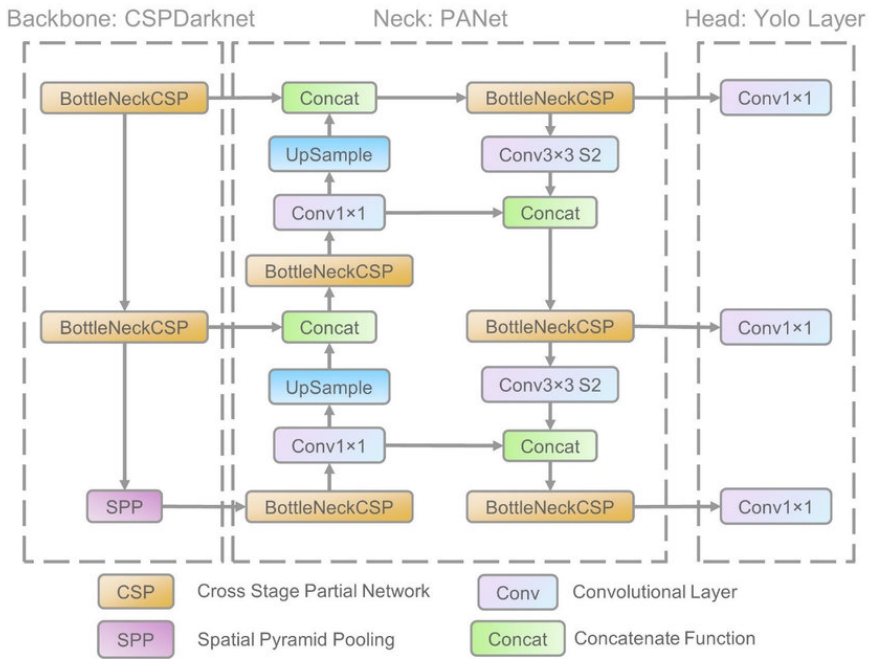

Figure 15. Simplified architecture of YOLOv5 model. Note: the v6 release of YOLOv5 replaced SPP layer with SPPF. Exact number of layers and convolutions depends on the selected model size [63].

introduced many novelties. Important to note that YOLOv5 is still under active development, and new sub-versions are regularly being released. In particular, authors decided to tag two subsequent major updates of the YOLOv5 model as v6 [22] and v7 [21] versions, respectively. Such naming introduced confusion in a YOLO model family line classification. In our work, we use the v6 release and base the further discussion on it. Hence, we emphasise that we will refer to the v6 release of the YOLOv5 model as just YOLOv5 if not mentioned otherwise.

The YOLOv5 model is comprised of backbone, neck and head networks as many other one-stage detectors. Although these sub-modules are similar to those used in previous YOLO versions, YOLOv5 incorporated changes to each of them to make the whole network faster and more accurate compared to predecessors. In addition, the authors introduced other training improvements like the elimination of grid sensitivity, new activation and loss functions, model scaling etc. Last, many techniques, like data augmentation, were taken from the previous YOLO versions. The simplified visualisation of YOLOv5 model architecture is shown in Figure 15.

Backbone. Cross Stage Partial (CSP) network strategy [60] was combined with Darknet53 backbone from YOLOv3 and that resulted in a new CSP-Darknet53 model architecture. CSP networks are designed to aid deep neural networks in overcoming the redundant gradients problem, which in turn occurs due to tackling the vanishing gradients problem. CSP network reduces the excessive amount of gradient information caused by dense layer connections by truncating the gradient flow. In particular, CSP partitions the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. That way, the information flow scope is preserved while computations are reduced. Figure 16 shows architecture modification by CSP connections.

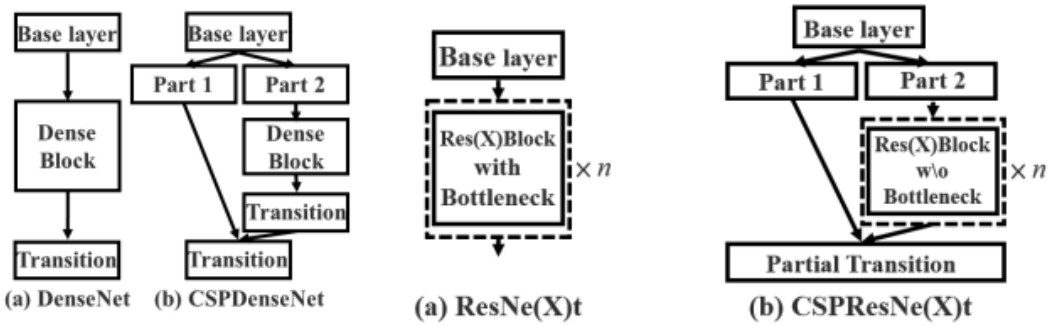

Figure 16. Applying CSP connections to DenseNet and ResNet architectures [60].

Neck. YOLOv5 uses a variant of Feature Pyramid Network (FPN) [30] called Path Aggregation Network (PANet) [33] which was adapted from YOLOv4. Just like in the

YOLOv5 backbone, CSP connections are also incorporated into PANet, which resulted in a new CSP-PANet architecture. In addition, the initial YOLOv5 version uses a Spatial Pyramid Pooling (SPP) layer on top of the very last layer of the backbone. It aggregates input information and returns a fixed-length output, greatly increasing the receptive field and segregating the most relevant context while having almost no effect on model speed. In the v6 version of YOLOv5, authors implemented a new SPPF architecture that achieves identical computations as SPP with fewer parameters.

Head. YOLOv5's head is identical to those used in YOLOv3 and YOLOv4. It comprises three convolutional layers used to predict bounding box dimension offsets t_x, t_y, t_w, t_h as well as objectness and class scores. However, formulas that derive native box coordinates space b_x, b_y, b_w, b_h have changed as shown in Equation 5, where c_x, c_y, p_w, p_h correspond to pairs of grid cell offsets and anchor box dimensions, respectively. YOLOv5 preserved the application of Non-Max Suppression (NMS) on top of the head predictions which was used from the very first YOLO model.

$$\begin{aligned} b_x &= (2 \cdot \sigma(t_x) - 0.5) + c_x \\ b_y &= (2 \cdot \sigma(t_y) - 0.5) + c_y \\ b_w &= p_w \cdot (2 \cdot \sigma(t_w))^2 \\ b_h &= p_h \cdot (2 \cdot \sigma(t_h))^2 \end{aligned} \tag{5}$$

Activation and loss functions. The authors chose a Sigmoid Linear Unit (SiLU, aka Swish activation) [43] activation function for all the hidden layers, which is smooth and differentiable at each point version of the ReLU function. YOLOv5 inherited CIoU loss for box regression from the YOLOv4 model. Lastly, objectness and classification scores are penalised by binary cross-entropy loss. Regarding the objectness loss, it is important to emphasise that the YOLOv5 network predicts the objectness score as an CIoU of ground truth and predicted box. In other words, the ground truth objectness value is not binary (i.e., 0 and 1 for object absence and presence, respectively) but continuous, representing how well the predicted box matches the ground truth object. Although the objectness loss penalty teaches the network to predict a correct CIoU without the aim of moving it toward 1, the coordinate loss forces the model to output a better box, which eventually pushes the CIoU to grow.

Eliminating grid sensitivity As explained in a Subsection 3.4.4, the YOLO model family produces dense predictions on the grid that are mapped back to the original input image space. Such segregation negatively impacts detection of objects that reside on the edge of 2 grid cell regions (or even worse in the case of 4 corners location). A model may struggle to assign a proper cell index for the input annotation. Hence, the authors eliminated such grid sensitivity by allowing neighbouring cells to predict objects

if they fall within the extended receptive region. In particular, the grid receptive field was expanded from [0; 1] to [-0.5, 1.5]. That means that close enough to the object center adjacent grid cells will be responsible for predicting objects in the neighbouring region (see Figure 17 for visual explanation). Important to note that this modification caused the change in b_x, b_y derivation formulas (Equation 5) in order to match a new offset range.

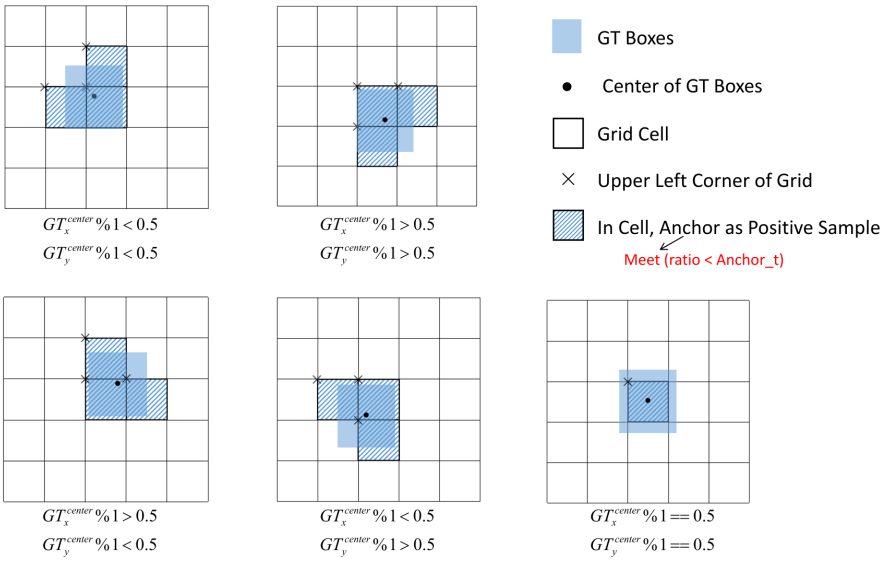

Figure 17. Grid cells assignment to an input ground truth (GT) box in YOLOv5 [22].

Data augmentation. Most of the data augmentation techniques were adapted from previous YOLO models. In particular, the most prominent are mosaic data augmentation [9] (stacking four images in a grid fashion to produce a new single input image), MixUp [68] (overlapping two images in a semi-transparent manner), CopyPaste [14] (pasting parts of one image into another). Other regular and more straightforward augmentation techniques like image axis flips, multi-scale resize, HSV colour space shifts, etc., are also present.

Model scaling. In the initial YOLOv5 release, authors provided pre-trained weights for 4 different model sizes: small, medium, large and extra large. Since the v6 release, they added a new nano model, which is incredibly small and is supposed to work fast on mobile CPUs. Model size scaling is achieved by compound shrinking/expanding the model's depth and width (number of feature maps per layer) using respective multiple coefficients. That approach is similar to EfficientDet solution [57], although YOLOv5 scaling does not automate image input size adjustment. In order to aid this, the v6 release

also incorporated new 5 models with larger default input sizes, although with the same nano, small, medium, large and extra large ranking.

3.5 Handling missing annotations in object detection

Missing annotations in object detection mean that data is only partially annotated, leaving some objects without labels. In other words, data mixes annotated and unannotated objects on an individual image level (see Figure 18). That phenomenon is sometimes referred to as a *sparse annotation* setting [61, 67]. Such a setting should not be confused with a partial case of a typical semi-supervised learning setup when the data comprises two sets: images with complete annotations and without annotations at all.

Figure 18. Example dataset with missing labels synthetically generated from PASCAL VOC dataset [4]. The green boxes are the ones present in the annotations. The red boxes are the ones that should have been annotated, but they are missing from the annotation. Each column represents from left to right the normal, easy, hard, and extreme settings [67].

The semi-supervised learning is an extension of regular supervised learning. It aims to discover more data to improve the performance of existing detectors. In contrast, a sparse annotation setting is an unavoidable constraint that makes the learning process less stable and transparent. In such a scenario, there is no obvious trustworthy way to

validate the object's presence in an unannotated image region. The only information that sometimes is known about the labelling quality is the approximate percentage of the annotated objects to the total of existing ones (overall in the dataset).

Missing annotation issue is inherent to many application domains. It frequently arises when the annotation process is time-consuming and/or challenging. For example, the high density of objects and expertise needed to perform labelling can become a significant obstacle to producing large high-quality datasets. Such a situation is widespread in the histopathology domain. Data comes with only a portion of objects being annotated, which makes neural network learning controversial because the network is taught inconsistently. In other words, objects that share the same features are simultaneously treated as positive (labelled) and negative (unlabelled) cases, which can result in unstable gradients.

A handful of research was conducted to address the issue of neural network training in missing annotation settings. Most of the primary solutions adopted various resampling-based methods to handle this problem. For example, Wu et al. [62] proposed a soft sampling technique to re-weight the gradients of object regions based on the overlaps with positive instances. Another work [42] suggested part-aware sampling that employs human intuition for establishing the hierarchical relation between labelled and unlabelled regions.

In a more recent study, Zhang et al. [67] moved further and proposed Background Recalibration Loss (BRL) to soften the penalty from unannotated objects. The idea was simple yet efficient. The authors suggested mirroring the focal loss function (explained in *Activation and loss functions* paragraph of Subsection 3.4.5) at the selected threshold value to lower the loss for confident false positives. They replaced the left-most part (up to threshold t) of the focal loss negative branch (i.e., the one that corresponds to the negative ground truth labels) with mirrored right-most part of the positive branch (i.e., the one that corresponds to the positive ground truth labels). The positive branch remains the same, but employs the poitive class weight factor α. The illustration of mirroring is shown in Figure 19 and the complete loss formula is in Equation 6. Thus, the new loss implicitly relabels negative samples as positive if the model confidently assumes a positive class. The authors showed that with their method, at least one annotated object

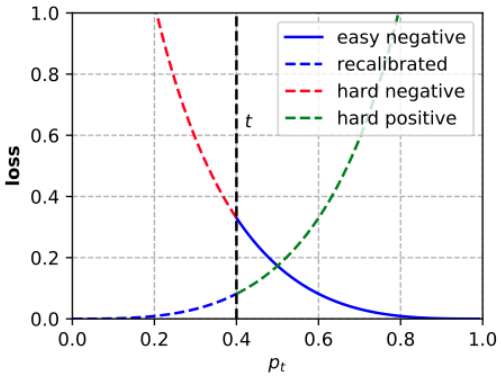

Figure 19. The plot of Background Recalibration Loss is denoted with blue colour. The p_t denotes negative class probability. The hard negative branch is replaced with mirrored easy positive branch up to the threshold `t` [67].

per image (in PASCAL VOC 2007 dataset [4]) is sufficient to prevent unstable gradients. Hence, they proved that efficient learning is feasible even under extreme annotation incompleteness.

$$\text{BRL}(y, p) = \begin{cases} -\alpha\,(1-p)^{\gamma} \log p, & \text{if } y = 1 \text{ or } y = 0 \text{ and } p > 1 - t \\ -p^{\gamma} \log(1-p), & \text{if } y = 0 \text{ and } p \leq 1 - t \end{cases} \tag{6}$$

An alternative solution of using the co-mining technique was proposed by Wang et al. [61]. The authors designed a Siamese network of two branches that send a positive supervision signal to each other to mine pseudo-labels. With that methodology, the authors reported 1.4% – 2.1% improvements compared to previous baselines. Although this approach provides a more substantial treatment of sparse annotations compared to previously published approaches [62, 42, 67], it comes with a cost of way more complex integration into existing frameworks due to significant model architecture changes.

4 Data and methods

This section first introduces two histology datasets used for our experiments. Then, it describes the methods we employed for training object detection models on this data. Most importantly, we describe solutions to aid training in missing annotation settings.

4.1 Datasets

Digital pathology is accompanied by a wide range of open-source datasets that can be used to train deep neural networks for supervised learning [8]. In our work, we employ two data sources for our experiments and performance tests. We select the publicly available MoNuSeg 2018 dataset [29, 28] for the purpose of model tuning and development. The dataset's versatility and abundance of annotations make it a powerful starting point for neural network training. Another dataset is private and is comprised of testis histology images and corresponding manual annotations. It is provided by the East Tallinn Central Hospital and incorporates intrinsically sparse and lower in volume annotation data.

We consider these datasets as instances of two different data provisioning scenarios. In the case of MoNuSeg 2018, it is polished, diverse, and high-quality data with an abundance of dense annotations. In contrast, testis data corresponds to real-world settings when the labelling budget is constrained, resulting in modest annotation volume with sparse labels. The subsequent subsections will introduce each dataset in more detail.

4.1.1 MoNuSeg 2018

We base our model training experiments on the MoNuSeg 2018 dataset [29, 28]. The subset of the data devoted for model training is comprised of 30 images of tissue samples extracted from 7 different organs, each of size 1000×1000 pixels. Each image is a X40 magnification sample from WSIs of individual patients downloaded from The Cancer Genomic Atlas [1]. These crops belong to separate unique WSIs, and the selection strategy was guided to choose the sub-regions with higher cell density. That was done to obtain maximum diversity while minimizing computational burden by creating a limited number of images.

Table 1. MoNuSeg 2018 training and testing datasets composition [41].

Subset	**Nuclei**	**Images**									
	Total	**Total**	**Breast**	**Liver**	**Kidney**	**Prostate**	**Bladder**	**Colon**	**Stomach**	**Lung**	**Brain**
Train	21 623	30	6	6	6	6	2	2	2	-	-
Test	7 223	14	2	-	3	2	2	1	-	2	2
Total	28 846	44	8	6	9	8	4	3	2	2	2

Training set images come with 21,623 hand-annotated nuclear boundaries. In addition, there is a test set comprised of 14 images that also include 7,223 manual annotations. Both train and test sets were prepared in the same way. The statistics of slides (including organ types) in the train and test sets are given in Table 1. Note that some organs are exclusive to the train and test sets. In MoNuSeg 2018 competition, such data split is supposed to promote generic models rather than those performing well only on training data.

The attractive property of the MoNuSeg 2018 dataset is its variability. Firstly, as it can be seen from the Table 1, it includes images of multiple organs, namely the kidney, lung, colon, breast, bladder, prostate, brain, stomach, and liver. That accounts for 9 organ types, although only 7 are used in both train and test sets. There is much diversity in organ appearance. For example, tissue colouring, cell morphology (size and shape), and cell density are notably different. All these can be concluded from Figure 20, which illustrates sample images from the test set. Secondly, both benign and malignant tissue samples were included. Finally, data came from many places. In particular, images were gathered from 18 different hospitals. Such acquisition introduced diverse staining practices and acquisition equipment (e.g., scanners). Hence, the inherent MoNuSeg dataset versatility makes it a good starting point for AI modelling.

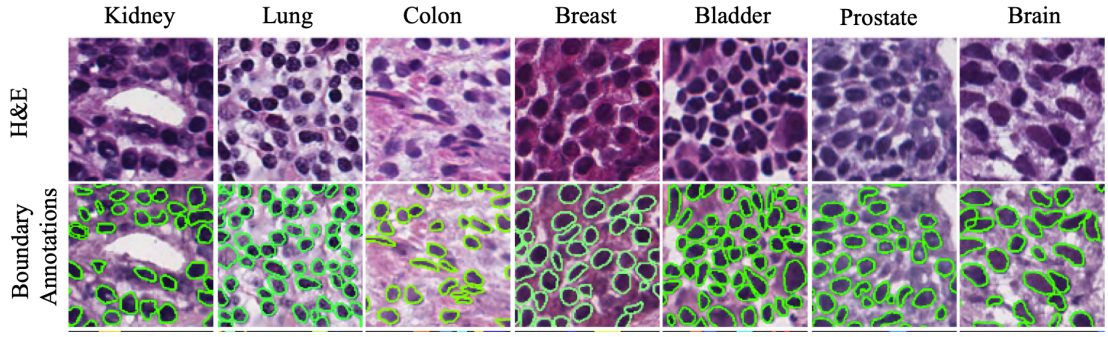

Figure 20. MoNuSeg 2018 test set sub-images taken from different organs. The first and the second rows show images with and without nuclei boundary annotations, respectively. Columns correspond to organ types [28].

4.1.2 Testis histology dataset

We obtained testis histology images from the East Tallinn Central Hospital. The dataset contains cells from five classes: spermatogonia, spermatocyte, spermatid, spermatozoa, and sertoly. Cells are annotated with bounding boxes and one of the five classes. Quantifying these cells in testis tissue is essential for studying spermatogenesis (germ cell development). Abnormalities in cell stages evolution can indicate spermatogenesis arrest, a phenomenon that causes men's infertility.

Table 2. Testis training and testing datasets composition.

Subset	# WSIs	# Crops	Cells					
			Total	Spermatogonia	Spermatocyte	Spermatid	Spermatozoa	Sertoly
Train	6	257	1468	336	383	353	80	334
Test	1	68	2325	907	691	489	-	238
Total	7	325	3793	1243	1074	842	80	572

The testis dataset consists of training and testing images. The training set comprises 6 WSIs, while the test set consists of 1 WSI. We selected a number of filtered patches (as described in Figure 3b of Subsection 3.1.2) of size 1024×1024 pixels from each slide at X40 magnification level. This resulted in 257 train and 68 test image patches. These sets contain 1486 and 2325 annotated cells, respectively. The summary of the testis data statistics is presented in Table 2.

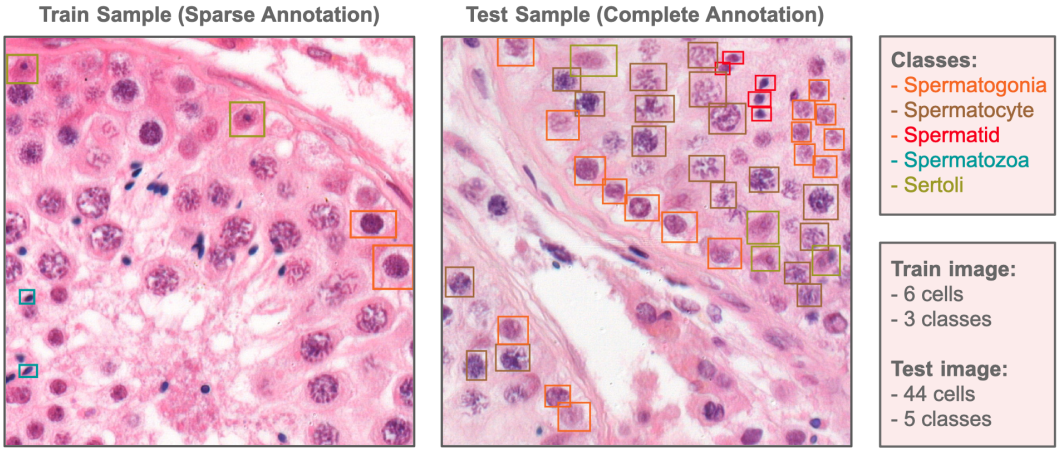

Figure 21. Sample train and test testis WSI crops at X40 magnification scale with overlaid cell annotations. The train set contains sparse annotations, while the test set has complete ones. The respective bounding box colour highlights cell class segregation.

Although the training data comprises a larger set of image crops than the test set (nearly four times more), the latter still contains nearly 60% more annotations than the former. Such a situation is conditioned by the inherently sparse training set, which means it lacks some portion of annotations. In contrast, the test set possesses complete annotations, which is essential to correctly estimate the model's training performance. Though, the spermatozoa class is missing from test images, so we omit it from the performance report. Sample tissue images from the train and test sets with overlaid cell annotations are shown in Figure 21.

It is important to emphasise that the cell counts in Table 2 refer to the number of annotated cells in the images. Consequently, the total count of cell objects present (including unannotated ones) in the training set significantly exceeds the reported numbers. Assuming that the distribution of cell frequency appearance in train and test samples is equal, we estimated that the train set includes nearly 8800 objects, out of which only 17% are annotated. Hence, we are dealing with significant annotation incompleteness.

4.2 Methods

Our work employs the YOLOv5 model [22] to train a cell detector in histology images. We chose this model as it provides both high detection speed and accuracy. We made several model adjustments to improve its performance further. Our development contributed to both improving the overall model detection rate (i.e., in complete annotation settings), but mainly focused on diminishing the accuracy drop due to missing annotations. As mentioned in Subsection 4.1, we employed the MoNuSeg 2018 dataset for performance estimates. The objective of obtaining better results on the test set guided our model tuning and development.

We performed simple model improvements like tweaking hyperparameters. But most importantly, we introduced more complex changes by customising the network loss function to achieve better performance under missing annotation constraints. In particular, we designed a new Generalised Background Recalibration Loss by adapting the Background Recalibration Loss [67] to the YOLOv5 model architecture, which required rethinking loss implementation. We describe model tuning and the new loss function in the subsequent sections.

4.2.1 Hyperparameter tuning

The YOLOv5 model has an extensive list of hyperparameters (listed in Appendix I) to tune the training process. We employed several techniques to find the best values. Since the hyperparameter space is too ample, it was only possible to partially automate the search. We used a combination of a grid search and a guided search. Grid search constructs a multitude of configurations to test the model by combining different values from the predefined ranges or sets. Since grid search is less efficient and requires much time to run, it was used for hyperparameters that have a less predictable impact on the training process (e.g., *optimiser*, its *learning rate* and *momentum*, learning rate *scheduler*). By guided search, we assume manual pick of configurations based on expected hyperparameter impact on our data. In other words, we integrated domain knowledge into our search decisions. For example, it is easier to figure out suitable data augmentation techniques because of their relatively transparent impact.

All the hyperparameters can be roughly separated into three main categories: architecture- (e.g., *model size*), optimisation- (e.g., *learning rate*) and data-related (e.g., *image*

flip). We adjusted these variables according to the best-performing configurations obtained during our search. The following paragraphs describe only variables that we changed as compared to the default values. The complete list of the default and our tuned hyperparameter values can be found in Appendix I.

Architecture-related. As described in *Model scaling* paragraph of Subsection 3.4.5, YOLOv5 encloses a family of models varied in size. Smaller models are faster but have limited learning capacity. However, bigger architectures are also prone to overfit. Since our image variability is low (300 images in the MoNuSeg 2018 training set), we decided to go with quite a limited model size. We chose a *small* model (YOLOv5s), which employs a 0.33 depth multiple ratio and 0.50 width multiple ratio, resulting in nearly 1.9M model parameters. In comparison, for the *large* configuration (YOLOv5l), both ratios are set to 1, and it has 46.5M parameters.

Optimisation-related. Optimisation parameters are crucial to make learning efficient. They allow for achieving the maximum performance within the given model learning capacity. Based on conducted hyperparameter search (including grid search), we lowered the *learning rate* (set *lr0* and *lrf* to 0.005 and 0.05, respectively) and increased *momentum* (set to 0.977). We also doubled the *batch size* up to 32 to speed up learning and make batch statistic calculations more representative during training. Additionally, we found enabling the *cosine* learning rate scheduler and learning with *image weights* (calculated internally) beneficial. Regarding the loss component, we slightly increased the objctness loss gain (contribution to the total loss coefficient) from 0.7 to 1. In some experiment setups (described in Subsection 5.2), we employed the change of obj_pw (positive class weight in objectness loss) hyperparameter from 1 to 0.1.

Data-related. The YOLOv5 offers a broad list of built-in data augmentation image transformations. However, our tuning tests showed that default values already provide the best configuration for our data. The only related hyperparameter value we changed is enabling *up-down* image flip augmentation with 0.5 probability. It is evident that histology images are invariant to any axis flip, which is a good example of domain knowledge integration in our model tuning process.

4.2.2 Generalised Background Recalibration Loss

Inspired by the Background Recalibration Loss discussed in Subsection 3.5, we designed a variation of this loss that is suitable for a broader set of object detection models (e.g., YOLOv5) and named it a Generalised Background Recalibration Loss (GBRL). The main obstacle that makes straightforward usage of BRL in models like YOLOv5

impossible is related to how ground truth probability scores are composed in different models. In more straightforward architectures, models learn to predict binary objectness scores (or conditioned class scores). In other words, ground truth negative and positive labels are equal to 0 and 1 probabilities, respectively, which was the assumption for the original BRL implementation. As described in *Activation and loss functions* paragraph of Subsection 3.4.5, YOLOv5 composes the ground truth objectness score as a CIoU metric between the ground truth and prediction boxes. Hence, ground truth values are no longer binary but continuous, ranging from 0 to 1. Such differences in model architectures can require the necessity of rethinking the implementation of the model loss.

As described in Subsection 3.5, BRL is based on a focal loss function. In our work, we employ $\gamma = 0$, which makes BRL a regular BCE loss. Hence, we onwards build our discussion based on BCE loss. The key feature of the BRL function is in the different treatment of positive and negative loss branches. The definition of these branches is straightforward when targets are binary. According to the Equation 2, positive and negative branches are defined as $-\log p$ and $-\log(1-p)$, respectively, when y equals 1 and 0 (See Figure 22a). However, with continuous y values, there is no evident loss segregation into positive and negative branches. The resulting loss is a combination of branches weighted by y and $1-y$, ranging from 0 to 1 (See Figure 22b).

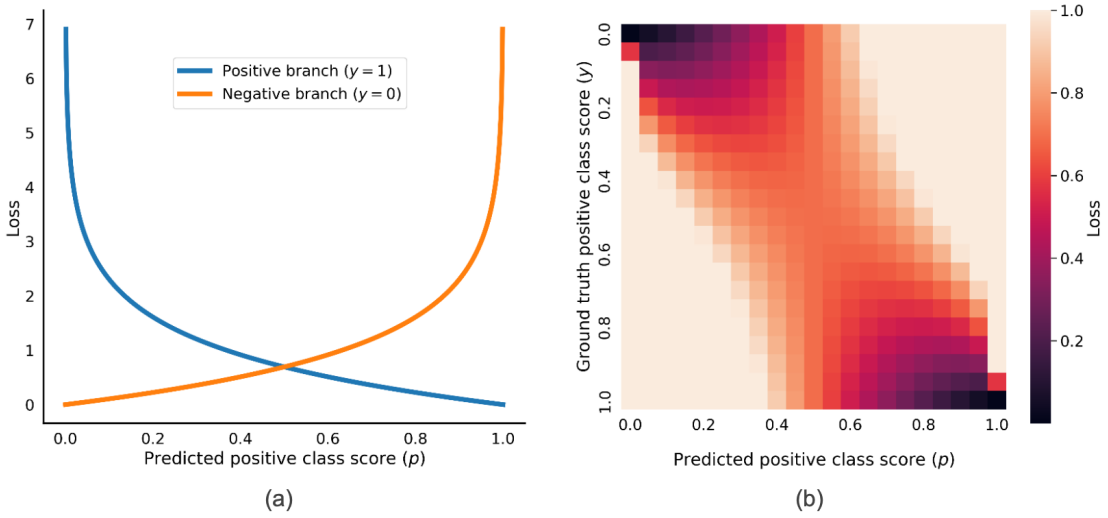

Figure 22. Figures *a* and *b* show BCE loss for binary and continuous ground truths, respectively. Figure *a* employs two line plots for respective y values, while Figure *b* shows a heatmap with continuous y range. In Figure *b*, a loss is truncated to a maximum value of 1 for better visualisation.

The main task for adapting BRL to continuous ground truth targets lay in rethinking the loss branch segregation concept. Since targets are built from a CIoU score, we

treat loss branches as follows. The negative branch corresponds to 0 CIoU because no object overlap means the absence of the ground truth object. Contrary, the positive branch corresponds to CIoU, which is strictly greater than 0 because any measure of objects overlap means the presence of the ground truth object. Then, we replace the *hard negative* part of the negative branch (up to threshold t) with the mirrored positive branch (i.e., *easy positive*). That way, if a model confidently predicts a positive class (i.e., over the threshold $1 - t$) for the negative ground truth label, our loss implicitly relabels the object to the positive instance. The GBRL loss is defined by the Equation 7. Here α is a positive class weight factor, which was shown to be important in missing annotation settings by Zhang et al.

$$\text{GBRL}(y, p) = \begin{cases} -\alpha\, y \log p - (1 - y) \log(1 - p), & \text{if } 0 < y \leq 1 \\ -\log(1 - p), & \text{if } y = 0 \text{ and } p \leq 1 - t \\ -\alpha \log(p), & \text{if } y = 0 \text{ and } p > 1 - t \end{cases} \tag{7}$$

With a new loss formula, we still can interpret the negative branch using a line plot as shown in Figure 23a. It now has less penalty for a confidently misclassified negative class, which should aid in missing annotation settings. The heatmap in Figure 23b shows loss values for the whole y range. Note how the upper row of the heatmap has changed compared to Figure 22b. This row now reflects the loss magnitude of the easy and hard negative branches shown in Figure 23*a*.

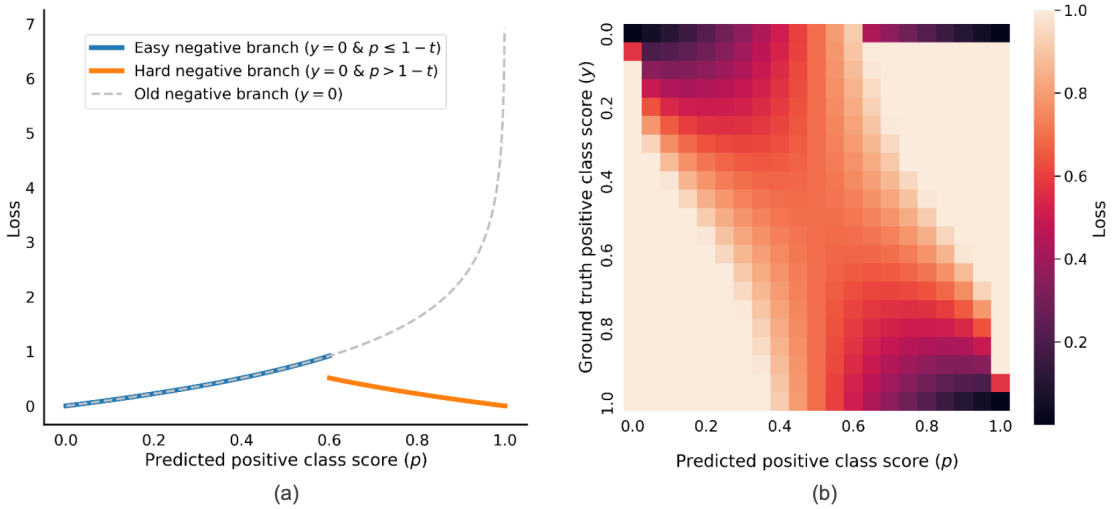

Figure 23. Generalised Background Recalibration Loss with threshold $t = 0.4$ and $\alpha = 1$. Figure *a* shows a new negative branch as a conjunction of the initial *easy* and changed *hard negative* subbranches while *b* shows a 2D heatmap for continuous y range. In Figure *b*, a loss is truncated to a maximum value of 1 for better visualisation.

5 Experiments and results

In this section, we describe training experiments conducted to test the methods proposed in Subsection 4.2. We first describe data preparation part, outline the experiment settings, and then present the results.

5.1 Data preparation

We followed a set of preprocessing operations documented by Nguyen et al. [41] to prepare MoNuSeg 2018 dataset [29, 28] for our experiments. First of all, the original MoNuSeg 2018 training set was separated into new training and validation sets of 25 and 5 images, respectively. Then, for initial 1000×1000 images, authors took 10 random sub-samples of size 512×512. That methodology was applied to all subsets, producing 250 training, 50 validation, and 140 testing images. Important to note that such an approach resulted in duplicate cell annotations because samples inevitably overlap. In particular, the whole dataset volume of annotations increased nearly three times. On the other side, image dimensions were reduced by almost a factor of 2, which made model training faster and less resource-consuming.

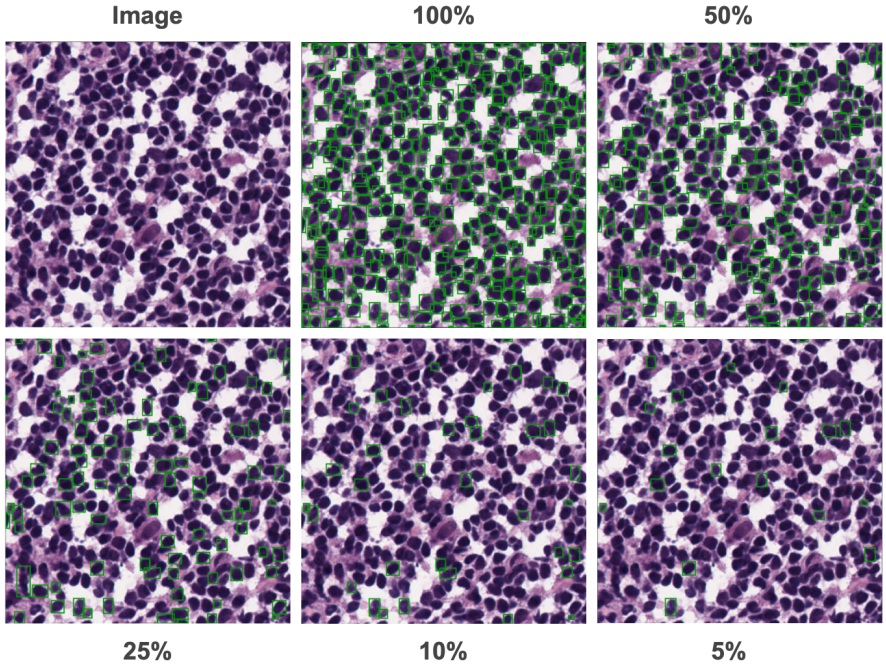

Figure 24. Annotation sampling strategy. Upper-left image shows a sample from MoNuSeg 2018 training set. Upper-centre image shows complete annotations. Other images illustrate sampled annotations with respective rates.

The next aspect of data preparation is a synthetic generation of annotation subsets. We uniformly sampled 50%, 25%, 10%, and 5% of annotations for each image in the training set to simulate incompleteness (illustrated in Figure 24). The validation and test sets remained complete for meaningful estimates. Important to note that the number of images in the whole dataset remained the same. In addition, we independently repeated random samplings five times (defined by random seeds) for each annotation percentage value to diminish the influence of lucky and unlucky seeds.

Regarding the histology testis dataset, there are no additional preprocessing stages in addition to those described in Subsection 4.1.2. The training set is intrinsically sparse (we estimated 83% of missing annotations), and the test set is complete.

5.2 Experiments

In our work, we employed several model training settings that varied in the combination of methodologies introduced in Subsection 4.2. We created five experiment settings to demonstrate an iterative performance gain (an ablation study). The following paragraphs describe each setup.

Setup 1: Model with no hyperparameter tuning. This is the most straightforward setup that involves no tuning of YOLOv5's hyperparameters. The only values we customised were the *batch size* (doubled from 16 to 32 to speed up training) and *image size* (to use the original 512×512 pixel resolution). The performance of this default configuration serves as a **baseline** benchmark to compare our methods.

Setup 2: Model with hyperparameter tuning. We applied hyperparameter tuning of the YOLOv5 model as described in Subsection 4.2.1, but left the obj_pw value with a default of 1. With this experiment, we planned to explore the performance potential of the existing model without changing its architecture, although we omitted the alpha factor adjustment to show its importance in further experiments.

Setup 3: Model with hyperparameter tuning and GBRL. In addition to the previous setup, we replaced the default BCE loss function of YOLOv5 with Generalised Background Recalibration Loss with parameters $t = 0.3$ (validated to give the best results) and $\alpha = 1$ (no weight shift, i.e., analogous to $obj\_pw = 1$). In this experiment, we estimated the benefit of introducing the GBRL.

Setup 4: Model with hyperparameter tuning and adjusted α factor. In addition to the second setup, we changed the objectness positive class weight factor obj_pw of the default YOLOv5 BCE loss to 0.1. In other words, we employ hyperparameter tuning described in Subsection 4.2.1 including the change of obj_pw. This benchmark

showcases the importance of calibrating the loss weight even without replacing the loss function.

Setup 5: Model with hyperparameter tuning, GBRL and adjusted α factor. Based on the third setup, but with α set to 0.1. This benchmark incorporates all the incremental improvements of the previous setups to establish the high accuracy gain in missing annotation settings.

In a context of MoNuSeg 2018 dataset, we followed the following steps for each of the five experiment settings. Firstly, we trained a single YOLOv5 model on the complete annotations to obtain a benchmark score. Secondly, we trained the YOLOv5 model for each sampling ratio and random seed, resulting in 20 independent models. We then averaged the performance metrics across different seeds to obtain representative estimations. Additionally, we calculated the standard deviation of scores to understand their variability. In result, we summarized the mAP_{50} scores on the test set for each experiment setup and missing annotation settings.

For the testis histology data, we used the baseline and best-performing models according to the results on MoNuSeg 2018 to estimate the performance on the test set. We preserved the model hyperparameter and loss modifications (described in Subsection 4.2) developed based on the MoNuSeg 2018 without any changes, except for the input image size and optimiser that we set to *Adam* instead of *SGD* (tested to work better).

5.3 Results

The following two subsection describe experiment results on MoNuSeg 2018 and testis histology datasets, respectively.

5.3.1 MoNuSeg 2018

The results of the experiments with the described setups are displayed in Table 3. Among all the highlights, we first want to emphasise the superior detection accuracy of YOLOv5 in complete annotation settings. The baseline model achieved a mAP_{50} score of 90.3%, and our best setup pushed it towards 90.9%. This is a significant improvement of 5.3% mAP_{50} compared to the results reported by Nguyen et al. The authors achieved a mAP_{50} score of 85.6% on the MoNuSeg 2018 test set with the dedicated CircleNet model [41].

Secondly, we would like to emphasise the intrinsic robustness of YOLOv5 to missing annotation settings. According to the baseline model, removing half of the annotations caused a drop of 1.1% in mAP_{50}. Going further to the extreme case of preserving only 5% of the annotations, the drop in performance was 6.7% in such a setting. Therefore, we can conclude that in this setup YOLOv5 is robust to sparsely annotated training data, indicating that large percentages of missing annotations lead to only small drops

in performance. In comparison, Zhang et al. reported a more significant accuracy drop when using RetinaNet [31] with a ResNet-101 [17] backbone over the PASCAL VOC 2007 dataset [4] with missing annotations. Although MoNuSeg 2018 and PASCAL VOC 2007 are very different datasets in terms of domains, it is worth noting that they possess similar amounts of annotations. Hence, the differences in results should be attributed to the applied models, although we cannot exclude the impact of the other factors.

Table 3. YOLOv5 object detection mAP_{50} scores on MoNuSeg 2018 test set obtained with different model setups and missing annotation settings. Note, **Alpha** term is used to denote α from Equation 7 and obj_pw hyperpararmeter of YOLOv5 in respective setups.

Experiment Setup				Data Subset				
№	**Tuned**	**GBRL**	**Alpha**	**100%**	**50%**	**25%**	**10%**	**5%**
1				*0.903*	0.892 ± 0.003	0.883 ± 0.015	0.870 ± 0.015	0.836 ± 0.030
2	✓			0.896	0.898 ± 0.005	0.890 ± 0.010	0.873 ± 0.018	0.840 ± 0.033
3	✓	✓		0.893	0.897 ± 0.010	0.888 ± 0.009	0.869 ± 0.022	0.838 ± 0.024
4	✓		✓	**0.909**	$\mathbf{0.907 \pm 0.002}$	0.901 ± 0.007	0.890 ± 0.012	0.847 ± 0.023
5	✓	✓	✓	0.906	0.905 ± 0.003	$\mathbf{0.903 \pm 0.004}$	$\mathbf{0.893 \pm 0.005}$	$\mathbf{0.868 \pm 0.010}$

Thirdly, we are comparing the performance across setups. According to Setup 2, our hyperparameter tuning (excluding obj_pw) brought minor improvements to detection accuracy on annotation subsets, except for the complete training set, which decreased accuracy. However, with an additional tuning of α (including obj_pw) in Setup 4, the model's performance substantially improved (see example detections in Figure 25), leading to the best results on the 100% and 50% annotation sets. Setup 3 shows that BRL without α tuning did not consistently improve performance compared to Setups 1 and 2. Lastly, and most importantly, Setup 5 demonstrates that combining all proposed features yields the best-performing model in highly sparse settings ($\leq 25\%$ annotations). Moreover, the standard deviation across the seeds was consistently reduced in Setups 4 and 5.

As a result, using Setup 5, we increased the mAP_{50} score on the 5%, 10%, and 25% of variants with the corresponding amount of annotated samples by 3.2%, 2.3%, and 2%, respectively, compared to the baseline Setup 1. Similarly, Setup 4 improved scores for 100% and 50% annotations by 1.5% and 0.6%. In this way, we further reduced the performance gap between the edge cases of annotation completeness (100% and 5%): there was a 3.8% gap in Setup 5 compared to 6.7% in the baseline. Interestingly, Setup 5 under 25% of annotations achieved the same score of 90.3% mAP_{50} as the baseline YOLOv5 model configuration in complete annotation settings.

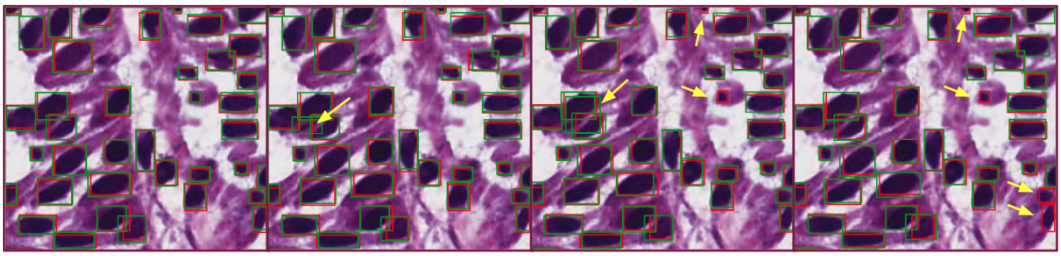

Figure 25. Example MoNuSeg 2018 test predictions of YOLOv5 models trained on datasets with randomly sampled annotations. Subset sampling ratio is displayed below the image (100% denotes no sampling). Red boxes correspond to ground truth, green ones — to predictions. Yellow arrows highlight false predictions.

Lastly, although we present the performance results of the GBRL under the complete annotations, its application is not reasonable in such a setup. Nevertheless, Zhang et al. showed that employing BRL did not significantly hurt performance while training with a fully annotated dataset. Surprisingly, in our experiments, GBLR with scaled alpha factor increased the mAP_{50} score slightly compared to the baseline.

5.3.2 Testis histology dataset

We got 44.4% and 50.7% mAP_{50} scores for the baseline (Setup 1) and best-performing in sparse settings (Setup 5) models, respectively, when analysing testis histology data. Employment of GBRL with *alpha* and hyperparameter tuning resulted in a 6.3% increase in scores. This is a high relative improvement, although it is important to note that the overall scores are lower for testis data than for MoNuSeg 2018, for a few reasons.

Firstly, the testis data incorporates a classification challenge with a high similarity of cell classes, making it difficult even for experts. Secondly, unlike in MoNuSeg 2018, the testis images contain cell objects that look very similar to the ones we are classifying, but do not belong to any of our target classes, and therefore they are not annotated. This led to frequent false positive confusion on these objects. Lastly, during the annotation process, cell class segregation depended on the higher-level information, such as cell location within the tubule. However, this information was often missing from the extracted image crops of WSIs at X40 magnification (i.e., the lack of a broader context), making inference of the proper class label challenging.

To test our assumption that the score decrease was due to the difficult classification problem, we conducted the same model training experiments but excluded the classification task. The results were 64.6% and 68.9% of mAP_{50} for Setup 1 and 5 models, respectively. Eliminating cell classification resulted in a significant score improvement of nearly 20% mAP_{50} for each model. The relative improvement between setups con-

stituted 4,3% of mAP_{50}. Figure 26 provides an example detections for each mentioned experiment on testis data, which clearly demonstrates the performance gain of our GBRL and model tuning solutions.

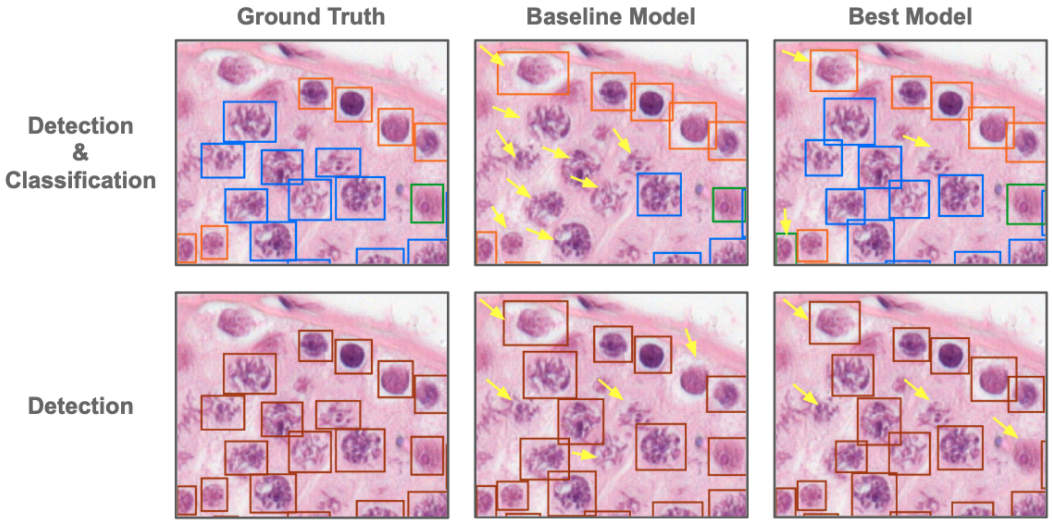

Figure 26. The YOLOv5 model trained on the incomplete annotations testis dataset predicted examples of testis images in the test set. The first column displays the ground truth boxes, the second column shows the predictions of the baseline model (Setup 1), and the third column shows the predictions of the best model (Setup 5). The first row shows models trained for detection and classification, while the second row shows detection only. In the first row, orange, blue, and green boxes correspond to the spermatogonia, spermatocyte, and sertoli classes. The yellow arrows indicate wrong predictions.

6 Conclusion

In this study, we investigated the training of object detection YOLOv5 model for histopathology in sparse annotation settings. Firstly, we demonstrated that hyperparameter tuning alone can enhance model performance. Thus, we increased the detection rate on MoNuSeg 2018 dataset over the previous state-of-the-art benchmarks [41] by 5.3% of mAP_{50}. We emphasised the importance of calibrating the objectness positive weight factor, denoted as α. This approach was most effective with complete and relatively high (50%) annotation completeness. Secondly, we proposed a novel Generalised Background Recalibration Loss (GBRL) function that extended previous works to be compatible with a broader range of models, including YOLOv5. When combined with α tuning, GBRL yielded a substantial performance gain in extremely sparse annotation scenarios with up to 95% missing labels. Our results showed that YOLOv5 trained on histopathology data was more robust to missing annotations than anticipated from prior work [67].

Sparse labelling can significantly reduce the time and budget costs associated with data preparation, which is especially critical in medical imaging. The outcomes of our work should facilitate the development of object detection models in histopathology domains using sparse data.

7 Acknowledgments

This work was funded by PerkinElmer, Inc. and the Estonian Ministry of Foreign Affairs Development Cooperation and Humanitarian Aid funds. The computational resources were provided by the High Performance Computing Center of the University of Tartu. I want to thank my supervisors, Mikhail Papkov and Dmytro Fishman, for their patient assistance and insightful feedback in accomplishing and writing this work. Also, I am grateful to Georgi Džaparidze and Erik Tamp, who prepared data for research purposes and consulted on histopathology domain-specific questions.

References

[1] The cancer genome atlas (tcga). http://cancergenome.nih.gov/.

[2] Grammarly: Writing ai assistance.

[3] Introducing chatgpt.

[4] The pascal visual object classes homepage. http://host.robots.ox.ac.uk/pascal/VOC/. Accessed: 2023-04-25.

[5] Precision-recall area under the curve (pr auc). https://www.stateoftheart.ai/concepts/a468a5b3-605a-4010-a8c5-2337b5275e43. Accessed: 2023-04-25.

[6] What is a neural network? https://www.tibco.com/reference-center/what-is-a-neural-network. Accessed: 2023-04-25.

[7] AmScope. Medicine and microscopes: How microscopes have impacted the healthcare field, September 2019. https://amscope.com/blogs/news/medicine-and-microscopes-how-microscopes-have-impacted-the-healthcare-field.

[8] Aïcha BenTaieb and Ghassan Hamarneh. Deep learning models for digital pathology, 2019.

[9] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[10] Manuel Carranza-García, Jesús Torres-Mateo, Pedro Lara-Benítez, and Jorge García-Gutiérrez. On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. *Remote Sensing*, 13:89, 12 2020.

[11] Angel Cruz-Roa, Ajay Basavanhally, Fabio González, Hannah Gilmore, Michael Feldman, Shridar Ganesan, Natalie Shih, John Tomaszewski, and Anant Madabhushi. Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. *Progress in Biomedical Optics and Imaging - Proceedings of SPIE*, 9041, 02 2014.

[12] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972.

[13] Max Ferguson, Ronay ak, Yung-Tsun Lee, and Kincho Law. Automatic localization of casting defects with convolutional neural networks. pages 1726–1735, 12 2017.

[14] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D. Cubuk, Quoc V. Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation, 2021.

[15] Ross Girshick. Fast r-cnn, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[19] Lee Jacobson. Introduction to artificial neural networks - part 1. `https://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7`, 2013. Accessed: 2023-04-25.

[20] Glenn Jocher, Liu Changyu, Adam Hogan, Lijun Yu , changyu98, Prashant Rai, and Trevor Sullivan. ultralytics/yolov5: Initial release, June 2020.

[21] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, November 2022.

[22] Glenn Jocher, Alex Stoken, Ayush Chaurasia, Jirka Borovec, NanoCode012, TaoXie, Yonghye Kwon, Kalen Michael, Liu Changyu, Jiacong Fang, Abhiram V, Laughing, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Jebastin Nadar, imyhxy, Lorenzo Mammana, AlexWang1900, Cristi Fati, Diego Montes, Jan Hajek, Laurentiu Diaconu, Mai Thanh Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106. ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support, October 2021.

[23] Dae-Young Kang, Pham Duong, and Jung-Chul Park. Application of deep learning in dentistry and implantology. *The Korean Academy of Oral and Maxillofacial Implantology*, 24:148–181, 09 2020.

[24] Vaibhav Khandelwal. The architecture and implementation of vgg-16. https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b. Accessed: 2023-04-25.

[25] Inho Kim, Kyungmin Kang, Youngjae Song, and Tae-Jung Kim. Application of artificial intelligence in pathology: Trends and challenges. *Diagnostics*, 12:2794, 11 2022.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[27] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks, 2014.

[28] Neeraj Kumar, Ruchika Verma, Deepak Anand, Yanning Zhou, Omer Fahri Onder, Efstratios Tsougenis, Hao Chen, Pheng-Ann Heng, Jiahui Li, Zhiqiang Hu, et al. A multi-organ nucleus segmentation challenge. *IEEE transactions on medical imaging*, 39(5):1380–1391, 2019.

[29] Neeraj Kumar, Ruchika Verma, Sanuj Sharma, Surabhi Bhargava, Abhishek Vahadane, and Amit Sethi. A dataset and a technique for generalized nuclear segmentation for computational pathology. *IEEE transactions on medical imaging*, 36(7):1550–1560, 2017.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[31] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[33] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation, 2018.

[34] Aditya Lohia, Kalyani Kadam, Rahul Joshi, and Arunkumar Bongale. Bibliometric analysis of one-stage and two-stage object detection. 02 2021.

[35] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[36] Cui M. and D.Y Zhang. Artificial intelligence and computational pathology, January 2021. https://doi.org/10.1038/s41374-020-00514-0.

[37] Salinas M, Rosas J, Iborra J, Manero H, and Pascual E. Revealing the precision of your manual cell counts. https://chemometec.com/perfect-precision-manual-cell-counting/#precision.

[38] Salinas M, Rosas J, Iborra J, Manero H, and Pascual E. Comparison of manual and automated cell counts in edta preserved synovial fluids. storage has little influence on the results, October 1997. https://pubmed.ncbi.nlm.nih.gov/9389224/.

[39] Michael T McCann, John A. Ozolek, Carlos A. Castro, Bahram Parvin, and Jelena Kovacevic. Automated histology analysis: Opportunities for signal processing. *IEEE Signal Processing Magazine*, 32(1):78–87, 2015.

[40] David S. McClintock, Jacob T. Abel, and Toby C. Cornish. *Whole Slide Imaging Hardware, Software, and Infrastructure*, pages 23–56. Springer International Publishing, Cham, 2022.

[41] Ethan H Nguyen, Haichun Yang, Ruining Deng, Yuzhe Lu, Zheyu Zhu, Joseph T Roland, Le Lu, Bennett A Landman, Agnes B Fogo, and Yuankai Huo. Circle representation for medical object detection. *IEEE transactions on medical imaging*, 41(3):746–754, 2021.

[42] Yusuke Niitani, Takuya Akiba, Tommi Kerola, Toru Ogawa, Shotaro Sano, and Shuji Suzuki. Sampling techniques for large-scale object detection from sparsely annotated objects, 2019.

[43] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[45] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[46] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[48] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression, 2019.

[49] Janosh Riebesell. Convolution operator. https://tikz.net/conv2d/. Accessed: 2023-04-25.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[51] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[52] Deval Shah. Mean average precision (map) explained: Everything you need to know. https://www.v7labs.com/blog/mean-average-precision, 2022. Accessed: 2023-04-25.

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[54] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[55] C Sommer, C Straehle, U Köthe, and F A Hamprecht. Ilastik: Interactive learning and segmentation toolkit. In *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 230–233, March 2011.

[56] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[57] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. 2019.

[58] Baxi V., Edwards R., and Montalto M. Digital pathology and artificial intelligence in translational medicine and clinical practice, October 2021. https://doi.org/10.1038/s41379-021-00919-2.

[59] Martha S Vokes and Anne E Carpenter. Using CellProfiler for automatic identification and measurement of biological objects in images, 2008.

[60] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn, 2019.

[61] Tiancai Wang, Tong Yang, Jiale Cao, and Xiangyu Zhang. Co-mining: Self-supervised learning for sparsely annotated object detection, 2021.

[62] Zhe Wu, Navaneeth Bodla, Bharat Singh, Mahyar Najibi, Rama Chellappa, and Larry S. Davis. Soft sampling for robust object detection, 2019.

[63] Renjie Xu, Haifeng Lin, Kangjie Lu, Lin Cao, and Yunfei Liu. A forest fire detection system based on ensemble learning. *Forests*, 12:217, 02 2021.

[64] Samir Yadav, Rahul Rathod, Sugat Pawar, Vaishali Pawar, and Sitaram More. Application of deep convulational neural network in medical image classification. 04 2021.

[65] Shivy Yohanandan. What is mean average precision (map) and how does it work. `https://xailient.com/blog/what-is-mean-average-precision-and-how-does-it-work/`, 2020. Accessed: 2023-04-25.

[66] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

[67] Han Zhang, Fangyi Chen, Zhiqiang Shen, Qiqi Hao, Chenchen Zhu, and Marios Savvides. Solving missing-annotation object detection with background recalibration loss. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1888–1892. IEEE, 2020.

[68] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.

[69] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression, 2019.

Appendix

I. Training details

Table 4 lists hyperparameters that control the training process of the YOLOv5 model [22]. The list includes a variable name, default value, changed (tuned) value, and description.

Table 4. Hyperparameters of YOLOv5. Changed parameters highlighted in **bold**.

Hyperparameter	Default value	Tuned value	Explanation
lr0	**0.01**	**0.005**	initial learning rate
lrf	**0.1**	**0.05**	final learning rate (lr0 * lrf)
momentum	**0.937**	**0.977**	SGD momentum/Adam beta1
weight_decay	0.0005	0.0005	optimiser weight decay
warmup_epochs	3.0	3.0	warmup epochs
warmup_momentum	0.8	0.8	warmup initial momentum
warmup_bias_lr	0.1	0.1	warmup initial bias lr
box	0.05	0.05	box regression loss weight
cls	0.3	0.3	classification loss weight
cls_pw	1.0	1.0	classification positive weight
obj	**0.7**	**1**	objectness loss gain
obj_pw	**1**	**1 or 0.1**	objectness positive weight
anchor_t	4.0	4.0	anchor-multiple threshold
fl_gamma	0.0	0.0	focal loss gamma
hsv_h	0.015	0.015	image HSV-Hue augmentation
hsv_s	0.7	0.7	image HSV-Saturation augmentation
hsv_v	0.4	0.4	image HSV-Value augmentation
degrees	0.0	0.0	image rotation (+/- deg)
translate	0.1	0.1	image translation (+/- fraction)
scale	0.9	0.9	image scale (+/- gain)
shear	0.0	0.0	image shear (+/- deg)
perspective	0.0	0.0	image perspective (+/- fraction)
flipud	**0**	**0.5**	image flip up-down (probability)
fliplr	0.5	0.5	image flip left-right (probability)
mosaic	1.0	1.0	image mosaic (probability)
mixup	0.1	0.1	image mixup (probability)
copy_paste	0.0	0.0	segment copy-paste (probability)
weights	YOLO-v5s.pt	YOLO-v5s.pt	pretrained model weights
epochs	300	300	number of training epochs
batch-size	**16**	**32**	batch size
imgsz	**640**	**512**	input image size (resize)
rect	disabled	disabled	rectangular training
noautoanchor	disabled	disabled	disable AutoAnchor algorithm
image-weights	**disabled**	**enabled**	weighted image selection for training
multi-scale	disabled	disabled	vary img-size +/- 50%
optimizer	SGD	SGD	neural network optimiser
cos-lr	**disabled**	**enabled**	cosine LR scheduler
patience	100	100	early stopping patience

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Denys Kaliuzhnyi**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Reducing the Effect of Incomplete Annotations in Object Detection for Histopathology,
   (title of thesis)

   supervised by Mikhail Papkov and Dmytro Fishman.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Denys Kaliuzhnyi
08/05/2023