SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 000

Naslov

Andi Škrgat

Zagreb, ožujak 2023.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Every year there are more software products coming into the football industry. Companies have great interest in entering the football market because of its great popularity and many potential clients. Most of the software used in football is based on computer vision solutions which are capable of generating useful information for football staff. Such products are used by head coaches, assistant coaches, youth coaches and football analysts all for the one goal of finding information capable of improving team's and player's performance. Software used in football can be divided into two main categories based on their main function.

One category of products supports mainly drawing visualizations and tagging specific events during the footage. Once is one such product that is very often used in Croatian National League. One of the core features Once software comes with is the possibility to tag actions based on the information about its end result. Most of the analysis software is also used in other sports. In this way, companies expand their market size and prove their usefulness on multiple plans.

StatsPerform distinguishes itself from other products since it is a vertical platform providing solutions for many sports and in many parts of each sport. Except for the tactical analysis, they are also providing betting information by modeling the probability of various sports events happening during the game. For the tactical analysis, the platform uses a concept of edge analysis where the goal is to identify important tactical insights based on the tracking data. One of the key features of edge analysis is team shape analysis. It allows analysts to identify the most likely formation of the next opponent, detect moments in the game when players make errors and open a space that can be exploited by the other team. The most advanced shape analysis feature allows linking shapes to actions' results so that the most useful team tshapes can be identified. Analysts can forward such information to players to better understand specific situations and use it in the next match to increase the chance of winning. More often than not, AI models are part of the software. For example, StatsPerform uses AI models in various ways to calculate each player's impact on the game. One such model

is Possession Value which measures player's impact has when doing on-ball actions on the probability of the team scoring in the next 10 seconds. Pattern analysis is used to identify the opponent's recurring passing chains so that teams could prepare and prevent the possibility of receiving a goal through such play. Measuring the pressure the player with the ball suffers from can be extremely valuable since it can be used to directly calculate the most optimal actions for each game moment. In the last few years, set-piece analysis has become very popular. England's national representation is a perfect example. During their journey toward World Cup 2018 semi-finals they scored eight out of eleven goals from a set-piece.

Slika 1.1: OptaVision makes use of human annotation and AI computer vision models. [15].

The team's manager, Gareth Southgate, said that most of the credit should go to the team's attacking coach Allan Russell who designed training sessions to exploit the opponent's weaknesses during set-pieces.

StatsPerform uses the data obtained from the system called Opta that generates data in real-time using a combination of AI models in computer vision and human annotation which makes the process of collecting data a human-in-the-loop system. Human involvement is necessary because as we will see later, both object detectors and object trackers aren't perfect and can fail in the process of collecting players' data.

All software products start their work by importing video footage. Once can use videos created with any camera and it converts it into footage ready to be used in the analysis process. Spiideo uses AI not just in the analysis process but also during the filming step, to obtain high-quality panoramic footages without human intervention.

This thesis mostly tries to imitate the system used by Opta and StatsPerform on a much smaller scale and show that it can still be useful for football staff. Chapter 2 provides necessary information one should know when working with computer vision systems

based on deep learning. Chapter 3 dives deep into the topic of object detection, describing differences between one-stage and two-stage detectors, explaining metrics that were used to evaluate detector's performance and provides reader with a theory that will help bridge the topic of object tracking. Chapter 4 deals with object tracking by first explaining how one object tracker could be implemented and then providing metrics to establish good understanding of what makes good tracker. The chapter dives deeply into the SORT family of object trackers since one of its models, StrongSORT, was employed for the tracking task.

As for the software, an application for football analysis has been implemented. The application visualizes results obtained from trained object detectors and tracker both in 3D and 2D space. Visualization in 2D space is possible thanks to a homography which allows us to discard objects detected in the scene that are not players or referee. The application has also built-in functionalities that allow user to correct mistakes created by imperfect object tracker. Finally, several statistics that coaches and analysts can use are implemented to prove the readiness of the software to be deployed in real industry scenarious.

2. Introduction to computer vision

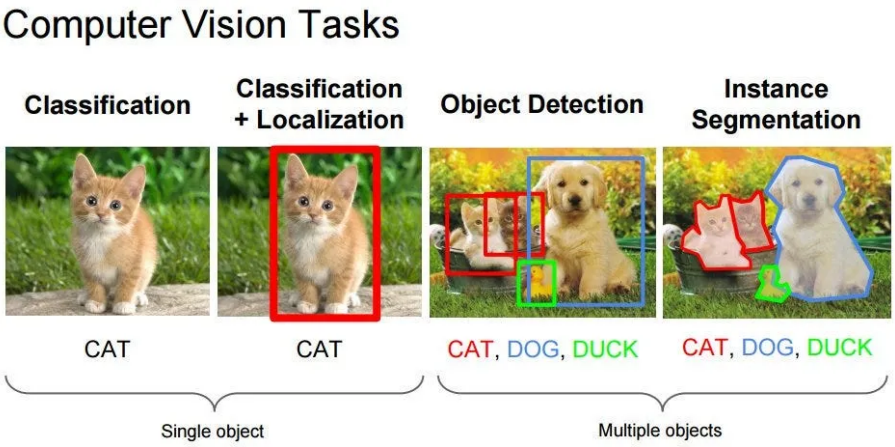
Computer vision is artificial intelligence's subcategory that provides theory for extracting information from visual inputs such as digital images and videos. Last 15 years were very fertile for computer vision. One of the most important reasons lie in the discovery of new optimization algorithms and deep learning models that can encode much more information more efficiently than before. Another important reason lies in the progress made on the hardware level. Graphical processing units (GPU) and Tensor processing units (TPU) can give us a significant performance boost thanks to the large number of cores and massive parallelism that can be exploited.

Long before exciting deep learning models and GPUs, people were interested in creating a system capable of describing objects in terms of their position and the action it does. One of the first noted experiments in the field of computer vision as we know it today was done in 1960. when a team of neurophysiologists concluded that image processing should first deal with simple shapes and then move to a more precise level. Their conclusion was based on observing cats' reactions upon showing them images with various levels of edges and lines. In the next 20 years, new algorithms capable of detecting edges and curves were discovered and the technology was already in a wide use. The discovery of convolutional neural networks by Kuniko Fukushima is an event that must be mentioned since it is extremely important for the further research progress made in next 30 years. From 2000.-2010. the technology was mostly used for creating real-time face recognition applications.

Back in past, the process of developing a new computer vision application was quite cumbersome because it required developers annotating and extracting features manually from images to use them in the detection. The first big improvement came with machine learning because features that previously had to be manually annotated, could now automatically be included in final calculations. Today, almost all researchers are focused on models based on deep learning since it can be used as a tool that automatically extracts and processes features in its pipeline without human intervention.

In 2023., computer vision can be found in a wide range of industries, starting from

medicine, sports and agriculture up to manufacturing, finance and retail. Depending on the use case, a specific method can be used but this thesis will deal with object detection and object tracking.



Slika 2.1: Typical tasks in computer vision [14].

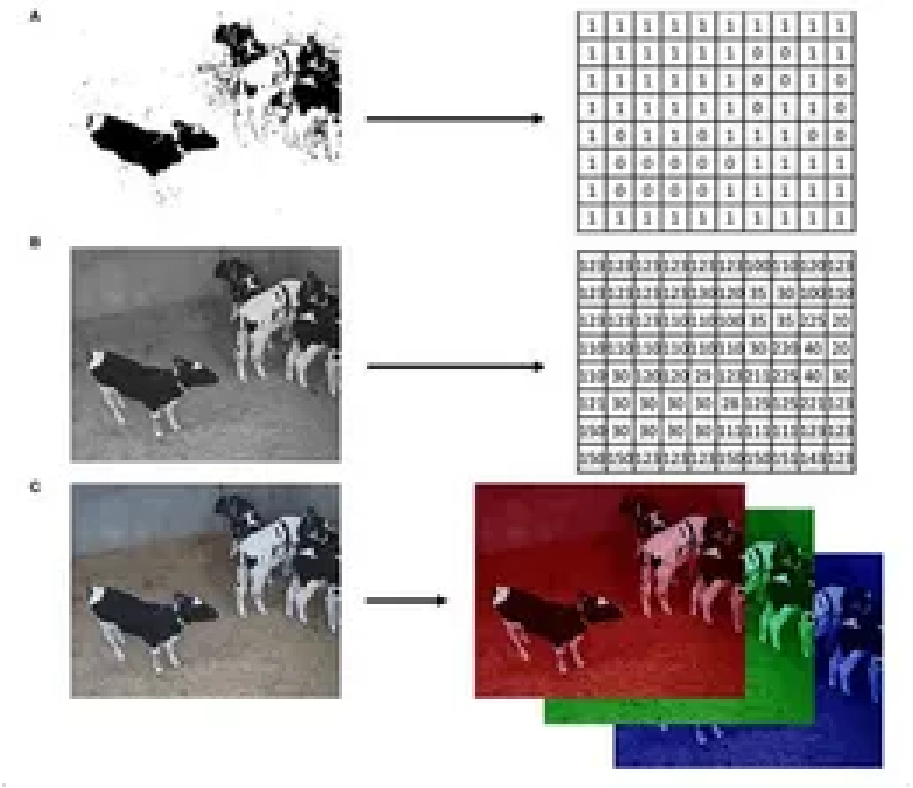
2.1. Image representation

Although humans find it intuitive understanding the image, computers need a thorough description of it. The convention is that image is represented as a 2D or 3D matrix where the first two dimensions represent height and width, while the third optional dimension is used if the image is coloured to represent the red, green and blue components of each image (RGB). Matrix is built from values called intensities and a number of bits used to represent the corresponding image pixel is called bit depth. Set of colours covered with the bit depth make gamut. If only one bit is used, the computer is dealing with binary images but usually, each value is encoded with 8 bits to get a set of 255 possible values for each pixel.

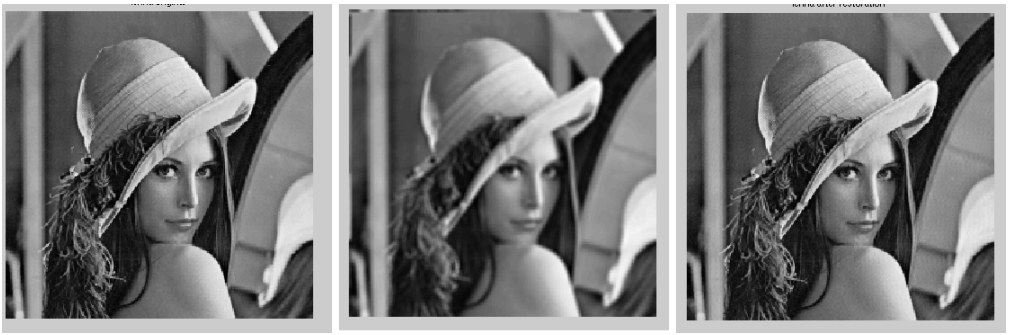
2.2. Image processing

Image processing is a term used to describe a set of techniques applied to the image once it is loaded into the process's memory. It consists of a sequence of numerical operations that are in some way useful for the application's final result.

Image restoration is a process of enhancing an image's quality by partially or completely removing noise, motion and blur from the existing, low-quality image. Image



Slika 2.2: Digital image representation [1].



Slika 2.3: On the left original image is shown, in the middle blurred and at the right restored image using inverse filtering [19].

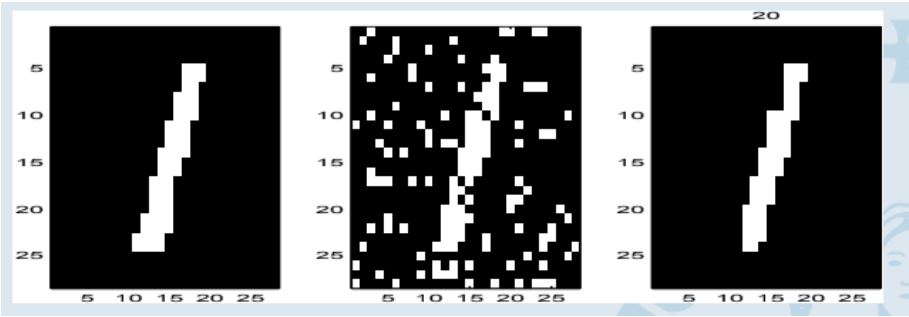
2.3 shows an example of applying inverse filtering for the sake of restoring image quality.

Markov networks are a probabilistic graphical model usually applied in image segmentation and image recovery. Let's imagine that are two images available to us, the original and the corrupted one where X represents clean pixels and Y corrupted pixels:

$$X = \{X_i, i = \{1, ..., D\}\}, X_i \in \{-1, 1\}$$

$$Y = \{Y_i, i = \{1, ..., D\}\}, Y_i \in \{-1, 1\}$$

Factor $(Y_i, X_i) = e^{X_i Y_i}$ is added so it would force corrupted pixels to be similar to "real" pixels and factor $(X_i, Y_i) = e^{X_i X_j}$ is used so that pixels would be as similar as possible and the final image smooth. These two factors are then combined to form a joint probability distribution $p(X, Y) \propto [\prod_{i=1}^{D} \phi(Y_i, X_i)][\prod_{i \sim j} \psi(X_i, Y_i)]$ that can be used for obtaining the most likely clean image from the set of corrupted pixels given as $p(X|Y)$.



Slika 2.4: Image restoration using Markov networks [16].

JPEG, JPEG2000, Huffman and LZW are examples of image compression algorithms whose purpose is to reduce the cost of storing images without sacrificing their quality.

Today, most of described processes are used as a preprocessing step for images entering the deep learning pipeline and aren't used alone since they lack expressivity. However, if applied correctly, procedures can significantly boost the performance of deep learning models.

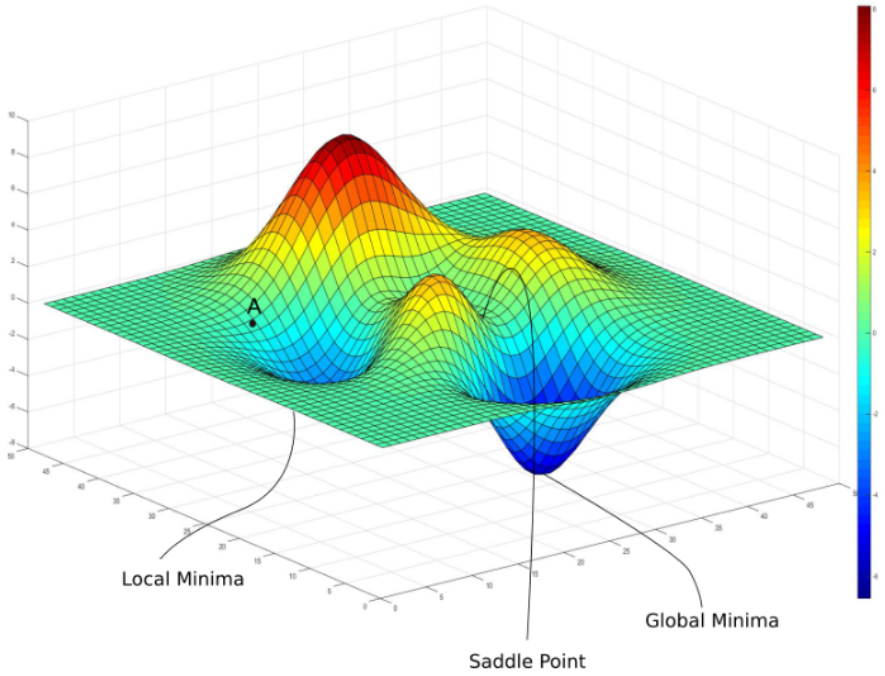
2.3. Deep learning in computer vision

Deep learning proved to be incredibly powerful in solving computer vision tasks. It's basic construct are deep neural networks, used to model complex structures like images. The word *deep* comes from a large number of layers deep neural networks use. Deep neural networks learn in total just one complicated function of the input data by combining outputs of each layer's learnt nonlinear transformation. As a mathematical model, neural networks are considered universal approximators because it is proven that for every possible set of input data, it is possible to find a two-layer neural network that can correctly explain continuous mapping $f(x) : [0, 1]^n \rightarrow R^m$. Hecht-Nielsen's

theorem proves this by combining results from Sprecher's and Kolmogorov's theorem. Sprecher's theorem definition will be skipped because of its complexity, while Kolmogorov's theorem simply states that any continuous function $f(x_1, ..., x_n)$ defined on $[0, 1]^n, n \geq 2$ is equivalent to:

$$f(x) = \sum_{j=1}^{2n+1} w_j (\sum_{i=1}^{n} \psi_{ij} x_i) \tag{2.1}$$

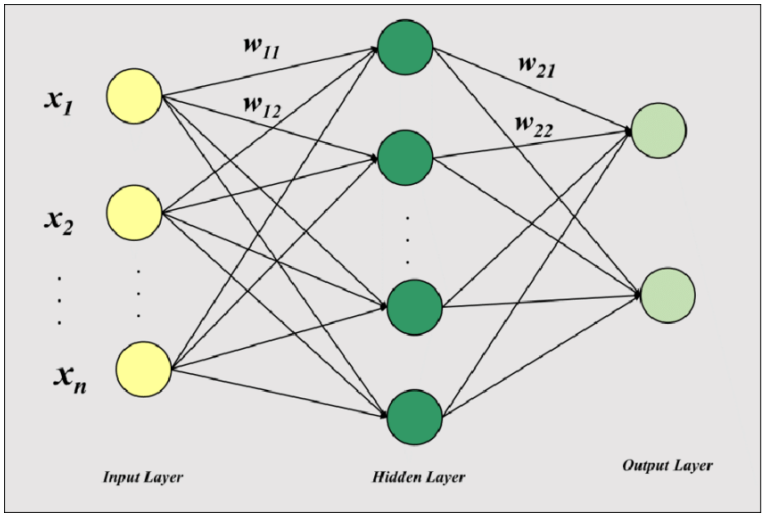
where functions w_j and ψ_{ij} are both continuous and ψ_{ij} is also monotone.
The core of the learning step in deep learning is gradient optimization. On the one hand, it is a great technique to perform optimization because it is super simple but on the other, a big problem occurs because of applying the convex optimization technique to the process of solving the non-convex optimization problem in high dimension. However, as research in deep learning progressed, it was empirically shown that this doesn't create big problems for deep neural networks.



Slika 2.5: Gradient optimization [7].

From the pure mathematical perspective, neural networks are nothing more than a large set of parameters tuned carefully and intelligently to provide output as close as possible to the one given in the training step. However, initializing the neural network's parameters proved to be an extremely important step since it is possible that with bad initialization gradients explode or become so small that learning effectively stops. Techniques that are most often used today are Xavier initialization and batch normaliza-

tion. Xavier initialization ensures that weights aren't too small or too big so that the learning signal could get propagated in the correct way. Batch normalization approaches the problem from a probabilistic aspect ensuring that the distribution of each layer stays the same.



Slika 2.6: Typical architecture of neural networks with one layer [21] that are mostly used for classification and regression.

Neural network optimization was already mentioned in several places so let's formally define it. Here it is assumed that the neural network solves the classification task but similar formulas can be derived for the regression tasks. Let's define X as a collection of inputs, Y as a collection of classes, W as the network's set of parameters, N as the number of input examples and let's impose a restriction that every input example must have exactly one assigned label. The learning procedure can be split into two passes, forward and backward. The forward pass consists of feeding every input into a neural network and at the network's end, obtaining a vector of numbers measuring the probability of the input example having a specific class. After probabilities have been obtained for all input examples, the final negative log-likelihood loss can be calculated as follows:

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} log(P(Y = Y_i|X_i)) \tag{2.2}$$

where in this case, Y_i is used to denote the example's correct class. In the backward pass, partial derivatives for each parameter are calculated and corresponding weights are changed in the direction of the steepest descent to minimize the final loss. A careful reader will notice the optimization is defined over all examples but in practice, the

batch of size B is used and parameters are optimized after passing through each batch. The loss from 2.2 then changes to:

$$L(W) = \frac{1}{B}\sum_{i=1}^{B} log(P(Y = Y_i|X_i)) \tag{2.3}$$

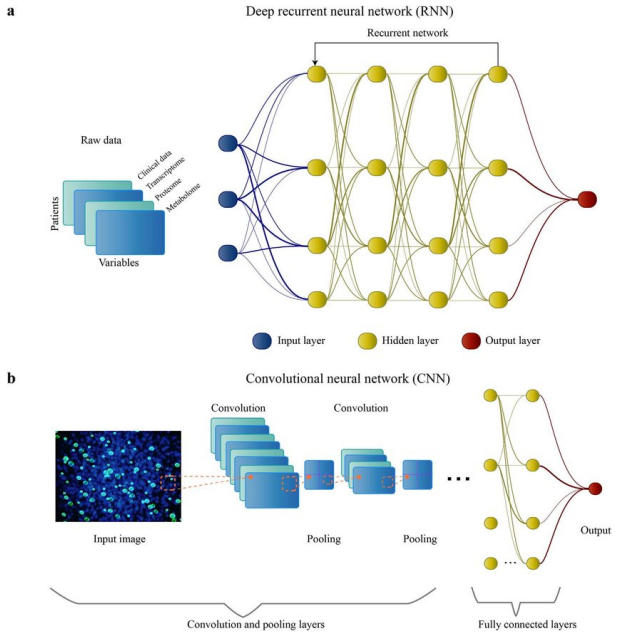
2.3.1. Convolutional neural networks

Convolutional neural networks are the most important neural network type in computer vision. As its name says, they use the mathematical operation called convolution in at least one of its layers. Convolution produces a function $s(t)$ from two real-valued functions $x(t)$ and $w(t)$ as following:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.4}$$

In the context of neural networks, the x function can be thought of as the input image, w as a kernel of a convolutional layer and s as a feature map. It is also worth saying that since images are discretized, instead of the integral, a summation over the image grid is used. Hence, the formula 2.4 can be modified to:

$$S(i,j) = (x * w)(i,j) = \sum_m \sum_n x(m,n)w(i-m, n-j) \tag{2.5}$$

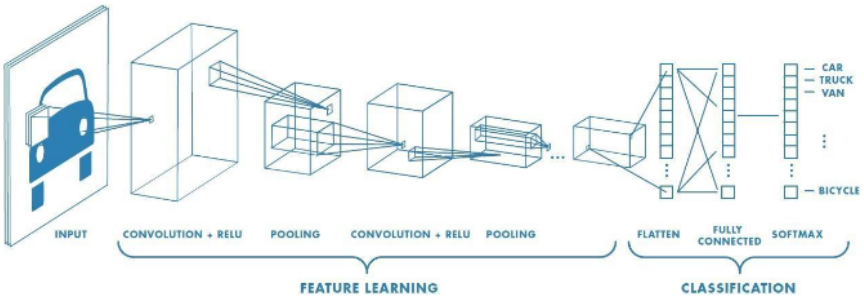


Slika 2.7: Architecture of deep neural networks [17].

Convolutional neural networks use two types of layers: convolutional and pooling ones. In general, such networks have several serious advantages over fully-connected neural networks. When dealing with images, taking care of memory is extremely important since it can easily explode. Convolutional networks use the so-called concept of sparse interactions so instead of defining a parameter for every possible combination of pixels in two consecutive layers, a kernel is defined that slides over the grid to detect useful features. From this point, it is easy to conclude that the same kernel is used on the whole image and this leads to a convolutional layer being translation equivariant or mathematically:

$$f(g(t)) = g(f(t)) \tag{2.6}$$

It is also shown that having such sparse connections and using kernels improves the model's statistical efficiency. The pooling layer has dual usefulness. Its most important function is reducing the element's neighbourhood to a single value so that output wouldn't be dependent on the exact feature's location in the image input. It does that by calculating various statistics like L2 norm, max, weighted and regular mean. In the literature, researchers often refer to this as the network's local invariance. The second very useful function pooling layer provides us with is the ability to downsample the image when dealing with the dataset where not all images are of the same size.



Slika 2.8: Convolutional neural network e2e [3].

Both pooling and convolutional layers have the notion of stride. The stride allows skipping some rows and columns from the input image so that better robustness could be achieved and some further computational efforts saved. Both layers can also use padding to preserve the image's input size after passing input through the layer. When no padding is used, the convolution is called valid. Sometimes, zero padding is added so that the output and input are of the same size and this type of convolution is called the same convolution. In the literature, full convolution can also be found, but it is very rarely used.

3. Object detection

Object detection is part of computer vision that locates instances on digital images and videos using deep learning models. Although there are many different definitions of object detection, the one that will be used in this thesis is that the result of the object detection algorithm applied to an image is a set of bounding boxes together with its assigned class label. This is important to mention because in many articles object detection is defined only as the process of generating bounding boxes or class labels. As described in section 2.2, there were many algorithms before the deep learning era. One of the most notorious algorithms that were used in object detection is based on background subtraction to detect an object as "a group of pixels". A variant of the Viola-Jones algorithm was also used together with the histogram of oriented gradients and scale-invariant feature transforms(SIFT) that were often combined with instance segmentation to get better understanding of occluded objects.

3.1. Evaluating the performance of object detectors

Every object detector needs to be evaluated using appropriate metrics. As mentioned, object detection deals with predicting both bounding boxes and class probabilities. Approximating bounding boxes relies on calculating loss. One of the most used loss types is MSE loss, defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{N}(Y_i - \hat{Y_i}) \tag{3.1}$$

where Y_i denotes the ground true value and $\hat{Y_i}$ denotes the predicted value. Intersection over union(IoU) is a computer vision equivalent to Jaccard similarity that encodes the quality of the predicted bounding box over the ground true bounding box.

$$IoU = \frac{area\ of\ overlap}{area\ of\ union} \tag{3.2}$$

A bounding box is considered to be a detection then if the IoU is over some threshold t.
From this point on, classic machine learning techniques would jump to the usage of accuracy. However, in object detection, this is impossible since it would require defining objects and parts of the image that the detector shouldn't to detect so calculations are moved towards precision and recall.

$$PR = \frac{\#TP}{\#TP + \#FP} \tag{3.3}$$

$$REC = \frac{\#TP}{\#TP + \#FN} \tag{3.4}$$

First, the precision-recall curve is drawn for all possible combinations and used for calculating average precision that can be described with the following formula:

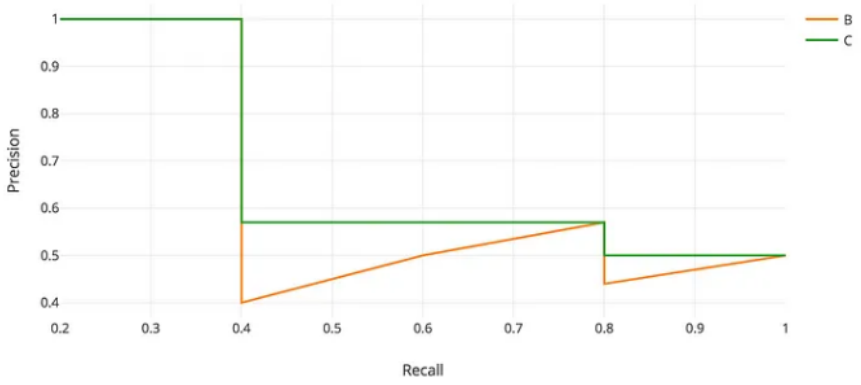
$$\begin{aligned} P_{interp}(R_{n+1}) &= max_{\tilde{R}:\tilde{R} \geq R_{n+1}} P(\tilde{R}) \\ AP &= \sum_{n} (R_{n+1} - R_n) P_{interp} R_{n+1} \end{aligned} \tag{3.5}$$

Average precision introduces the concept of interpolated precision which can be defined as maximum precision for all recall values larger than the recall value paired with the interpolated precision. In practice, mean average precision is used that just takes average over all classes.

$$mAP = \sum_{i}^{C} AP_i \tag{3.6}$$

Some models prefer using AP11 that interpolates precision over eleven equally spaced values. Sometimes AP also gets a suffix denoting the threshold value used for IoU calculations so for example, AP50 means a bounding box is an object if IoU is $\geq$ than 50%, AP75 for 75% etc. In AP@50:5.95, thresholds are used starting from 0.5 up to 0.95 with a step of 0.05. AP Across Scales metric tries to be as objective as possible by classifying objects into three categories by size; small, medium and large. Note by using recall instead of precision, a new set of metrics would be obtained like Average Recall, Average Recall Across Scales etc.

Generalized Intersection over Union (GIoU) tries to address the problem of being invariant to the scale of an object. It also solves the problem of vanishing gradients in non-overlapping cases but slowly leads towards poor regression results. It can be defined as:



Slika 3.1: Mean average precision in object detection [4].

$$GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|} \tag{3.7}$$

where C is the smallest convex object enclosing both A and B.
Distance loss, inspired by GIoU, also successfully solves the problem of vanishing gradients on non-overlapping cases but converges much faster. It can be defined with the formula:

$$DIoU = 1 - IoU + \frac{d^2}{C^2} \tag{3.8}$$

where C is the length of the enclosing box's, as defined with GIoU, and d is the distance between centers of predicted and ground true bounding boxes.
Complete Intersection over Union (CIoU) is the most powerful loss function that is applied in YOLOv3, YOLOv4, SSD and Faster R-CNN models. It is called complete because it aggregates information from overlap, aspect ratio and distance between predicted and ground true bounding boxes in an unique fashion. It is defined as:

$$L = S(B, B^{gt}) + D(B, B^{gt}) + V(B, B^{gt}) \tag{3.9}$$

where S is reversely proportional to the IoU, D is defined as the distance between two center points and V represents the consistency of the aspect ratio. For the end of this section, let's just briefly mention that when objects are applied to real-time videos, number of frames that model is capable of processing in a second is extremely important measure. In such use cases, one-stage object detectors are the way to go.

3.2. One stage and two stage object detectors

Today almost all object detection models are based on deep learning. Such models need to solve two basic tasks:

1. In the detection step find all objects occurring in the image.
2. In the identification step estimate bounding boxes and classify objects into one of the possible input classes.

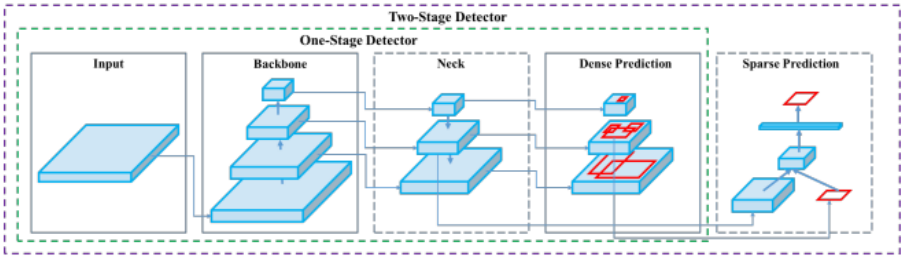
Based on whether those two tasks are done completely apart from each other or merged into one big task, two types of detectors emerged, one-stage and two-stage object detectors.

3.2.1. Two-stage object detectors

Two-stage object detectors usually use deep models for the detection part. The result of the detection part are so-called region proposals which are then fed to the identification step. Before diving into the algorithms used in the detection part of some famous two-stage object detectors, let's explain how would the brute-force solution look on the example of sliding window algorithm. The algorithm "slides" over whole image grid at different scales, crops the part of the image and passes it through the convolutional layer to obtain useful features. Sliding over the whole image is bad enough and iterating over different scales further increases computational cost so it is easy to see why this method is not used in the industry. The expensiveness of the method can be reduced with the usage of a larger sliding window and strides but this leads to poor object localization.
Region-based convolutional neural network (R-CNN) is one of the most important representatives of two-stage object detectors. In the detection part, it uses selective search algorithm, a representative of the region proposal algorithms, instead of a sliding window algorithm. Region proposal algorithms take into account some of the image's features like colour or texture and identify regions that could contain objects. The goal of such algorithms is to generate regions which will contain as many candidates as possible, thus they can be labelled as high-recall algorithms.

R-CNN finishes the work by further improving the precision of bounding boxes and predicting classes. Although R-CNN performs very well in practice, the selective search algorithm generates many region proposals which negatively influence performance.

Slika 3.2: Selective search in object recognition [18].



Slika 3.3: One-stage vs. two-stage object detectors [2].

Fast R-CNN tries to address the computational expensiveness of the R-CNN model by dealing with a single multi-task loss that combines losses coming from the wrong class predictions and inaccurate bounding boxes in the following way:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u >= 1]L_{loc}(t^u, v) \tag{3.10}$$

where L_{cls} represents a classification loss and L_{loc} represents regression loss. The Fast R-CNN model also introduces hierarchical batch sampling to speed up the computation. The method samples N images and then R/N from regions of interest (RoI) from each image. It was empirically shown that this works much better than sampling one RoI from R images.

Faster R-CNN is a model that uses a region proposal network (RPN) to obtain region proposals in an extremely efficient way. Its efficiency is based on the convolutional features that are shared with the detection network. RPN can also be merged with the detection network to obtain a single, end-to-end trainable deep neural network, with the usage of an attention mechanism.

Algoritam 1 Selective search method for generating region proposals.

Input:

I = input image

Output:

List[Image] = list of region proposals

CalculateSimilarity(r_i, r_j):

$color\_sim = s_{color}(r_i, r_j) = \sum_{k=1}^{n} min(c_i^k, c_j^k)$ ▷ $c =$ color descriptor

$text\_sim = s_{texture}(r_i, r_j) = \sum_{k=1}^{n} min(t_i^k, t_j^k)$ ▷ $t =$ texture descriptor

$size\_sim = s_{size}(r_i, r_j) = 1 - \frac{size(r_i)+size(r_j)}{size(im)}$ ▷ $size(im) =$ size of image

$shape\_sim = s_{shape}(r_i, r_j) = 1 - \frac{size(BB_{ij})-size(r_i)-size(r_j)}{size(im)}$ ▷ $size(BB_{ij}) =$ bounding box size around r_i and r_j

$final\_sim = a_1 * color\_sim + a_2 * text\_sim + a_3 * size\_sim + a_4 * shape\_sim$

Main:

over_segmented_img = OverSegment(I) ▷ Graph-based segmentaton method

merging = True

region_proposals = []

bboxes = []

while merging **do**

  for $i \leftarrow 1$ **to** $size(region_proposals)$ **do**

    for $j \leftarrow 1$ **to** $size(region_proposals)$ **do**

      sim = CalculateSimilarity(r_i, r_j)

      if sim > THRESHOLD **then**

        AppendBox(r_i, r_j)

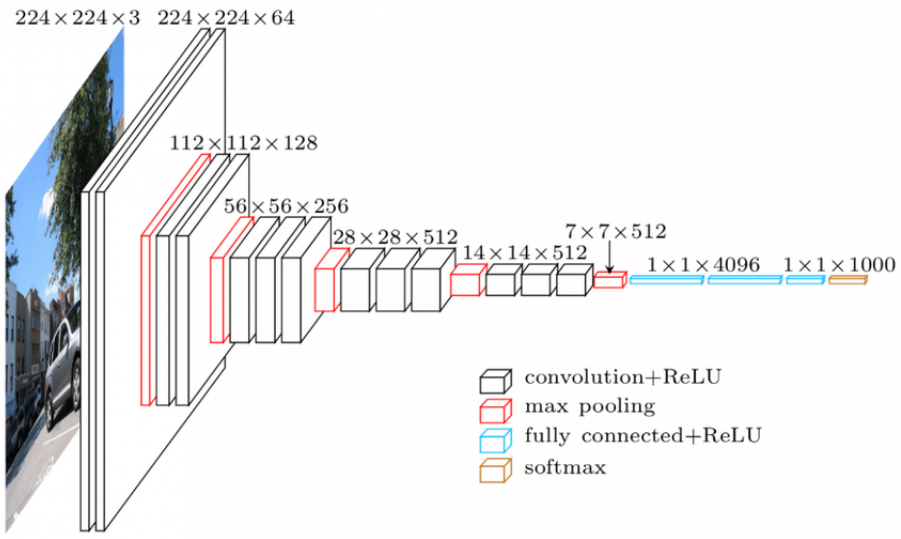
      end if

Mask R-CNN extends on the existing Faster R-CNN model by adding the execution branch that predicts the object mask together with bounding boxes providing us with a method for doing simple object instance segmentation with the usage of only one neural network.

3.3. One-stage object detectors

Two-stage object detectors usually are more accurate than one-stage detectors but are slower and hence, not very likeable in real-time applications. One-stage object detectors start with building a feature representation from the original image by using the

backbone network. The backbone network is a neural network that is almost always pre-trained on a large dataset (e.g ImageNet) to output rich and meaningful feature representation from new images without requiring additional training.



Slika 3.4: VGG backbone network [10].

Image 3.4 shows the VGG backbone network that outputs a vector of dimensionality 1000 after applying a softmax function on an output of a fully connected layer. However, one-stage object detectors don't need an output from the softmax function so instead, they work with, using again the image 3.4, $7 * 7 * 512$ feature maps. This is important because in this way it is possible to map these 7*7 grid cells to the original image like shown on the image 3.5

The core idea in single-stage object detectors is that for each object detected in the image, there is one main grid cell that will get activated by using an additional convolutional layer which combines feature maps from the backbone network. Those grid cells that contain an object also need to know what is the probability that the object is in its grid cell, of which class the object is and what are bounding box coordinates around it which means that the layer needs to produce $5 + C$ output channels for each grid cell in the image. Lastly, since every grid cell can be responsible for more than one object, expression $(5 + C)$ needs to be multiplied by the fixed parameter B.
Some of the best-performing single-stage detectors are part of the You Only Live Once(YOLO) family, Single Shot Detection (SSD), SquuezeDet and DetectNet. A detailed explanation of YOLO models will be provided in this thesis since they were used in the implementation.

Slika 3.5: Feature maps can be related with the original image [5].

3.4. YOLO models

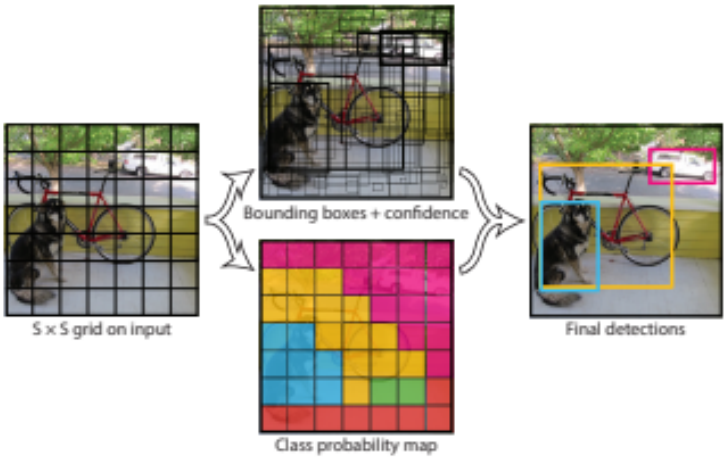
The first YOLO model came as one of the first object detector models that could run in real-time, offering close to 45 FPS when ran on videos. However, due to its simple architecture, it comes with the serious disadvantage of being able to detect a limited number of objects and often failing to detect very small objects.

3.4.1. YOLOv1

The original YOLO model uses the GoogLeNet with 24 convolutional layers and two fully-connected layers as the backbone network to obtain feature maps of shape $7 * 7 * 30$.

In later versions, the network was changed into DarkNet-19. The YOLO model uses the procedure in the same way as described in the chapter 3.3 so it is trained to directly predict bounding box coordinates. Hence, it also outputs a probability vector of size B for each grid cell. Using p_{obj} as the probability that the object is in the grid cell and IoU denoting the measure of intersection between the labelled bounding box and the predicted, each element of the probability vector is calculated using the following formula:

$$b_i = p_{obj} * IoU(pred, ground\_truth) \tag{3.11}$$



Slika 3.6: YOLO's algorithm explanation [13].

where p_{obj} approximates the intersection over the union between the ground truth bounding box and the predicted one.

Since object detection is usually performed on multiple classes and the first version of the model didn't support multiple classes in one grid cell, each element is further multiplied with the conditional probability $p(c_i|object)$ that gives us the final probability vector composed of elements calculated as:

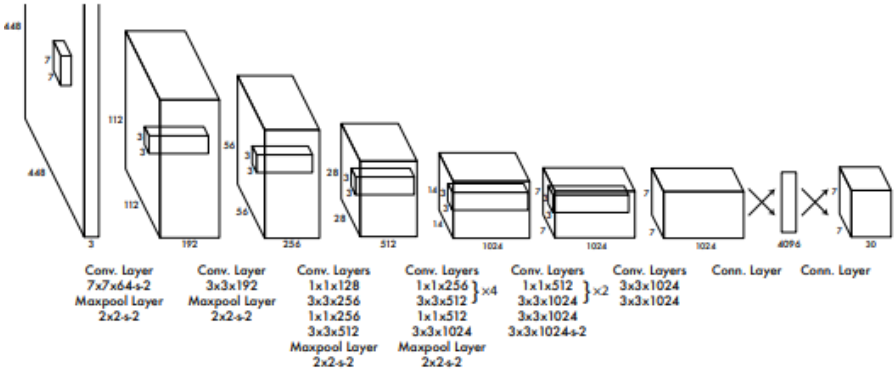
$$p_i = p(c_i|object) * b_i \tag{3.12}$$

The bounding box with the highest p_i is then proclaimed as an object.
All layers in the YOLO model, use the leaky rectified linear activation unit(Leaky ReLU):

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

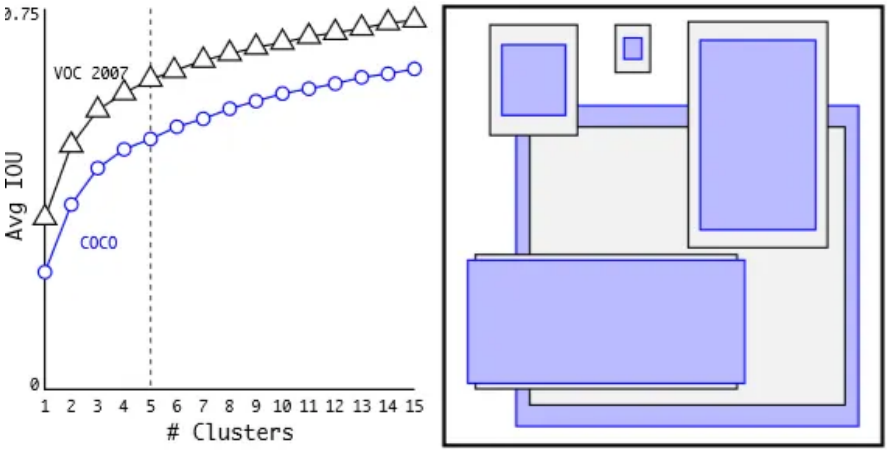
3.4.2. YOLOv2

Although an extremely powerful algorithm, YOLO has a relatively low recall because of failing to detect small objects and high localization errors. YOLOv2 introduces several novelties to improve YOLO's shortcomings. Except for adding batch normalization in every convolutional layer, YOLOv2 also performs fine-tuning of the DarkNet-19, the new backbone network, for 10 epochs on the ImageNet dataset before using it in the detection step. Fine-tuning is performed on images of size $448 * 448$ so the network could adapt to the image size that will be used in the detection step because



Slika 3.7: YOLOv1 arrchitecture [13].

it was shown that this was creating problems in YOLO. Instead of directly predicting multiple bounding boxes as in the first version, YOLOv2 introduces a concept of anchor boxes. They can be understood as prior guesses of what the size of real detections could be. If the user knows the detection size, determining the size of anchor boxes is easy. However, more often than not, detections occur at multiple scales so YOLOv2 at the beginning performs a k-means clustering algorithm to get a few centroids that will best describe the size of detections in the original dataset.



Slika 3.8: Choosing most informative number of clusters on VOC and COOC dataset [6].

By running a clustering algorithm at the beginning, it also captures the scale of all classes in the dataset without sacrificing precision. It is also trivial to see that, as better prior guesses of the detection size are, the better performance will YOLOv2 model have because there will be a smaller error when calculating displacements. By using anchor boxes, YOLOv2 solves the easier task of resizing the anchor box to the detection size than predicting the correct coordinates of the bounding box. A subtle

issue that is avoided with the introduction of anchor boxes is the model favouring larger bounding boxes. Last but not least, anchor boxes lead to smaller localization errors. Concrete coordinates of bounding boxes are then calculated as follows:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{3.13}$$

where c_x and c_y are cell's offset from the top-left corner, p_w and p_h are anchor box's dimension and t_x, t_y, t_w and t_h are network's output.

3.4.3. YOLOv3

YOLOv3 introduces a concept of objectness score. Each ground truth has at most one assigned bounding box prior and that is the one with the highest intersection over union. Objectness score is predicted using logistic regression and it has value 1 for a bounding box prior that overlaps the ground truth object more than any other prior bounding box. YOLOv3 allow bounding boxes to represent more than one class. This has a consequence on the training because class predictions aren't calculated anymore using the softmax, but rather using independent logistic classifiers.

The model switched from Darknet-19 to Darknet-53 for a backbone network. The network has 53 convolutional layers, has some shortcut connections as in classic residual networks and at the moment of introduction, had the highest measured floating point operations per second because of better GPU utilization. K-means clustering is still performed at the beginning but instead of five, as in YOLOv2, this model uses three different sizes for prior guesses. Contrary to its predecessors, YOLOv3 performs better in detecting smaller objects than medium or larger objects.

3.4.4. YOLOv4

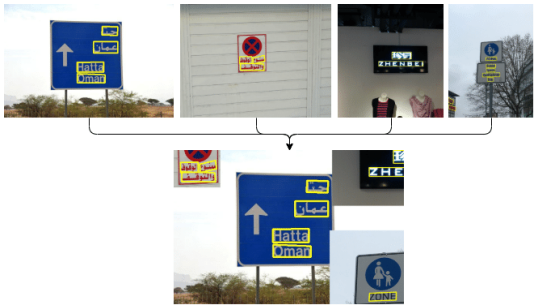
YOLOv4 makes use of several methods that don't necessarily need to be used in object detection. When a method influences only training cost or training modus operandi, it is considered a bag of freebies technique and so it especially makes sense to apply it in offline applications.

An example of a bag of freebies technique is data augmentation. Data augmentation encapsulates photometric and geometric distortions that can both be described as

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

Slika 3.9: DarkNet-53 architecture with 53 convolutional layers [12].

pixel-wise adjustment techniques that improve the model's robustness. Geometric distortions assume scaling, cropping, flipping or rotating the image while photometric distortions change the image's saturation, contrast, hue or brightness. There is also a specific type of mosaic data augmentation that improves feature identification on smaller scales by combining images from the training set in different ratios. Self-adversarial training is done in two steps. In the first step, some set of image's features are changed so that neural network would fail to detect the object. In the second step, network's parameters are updated so that next time, the network could successfully with deal with a similar example.

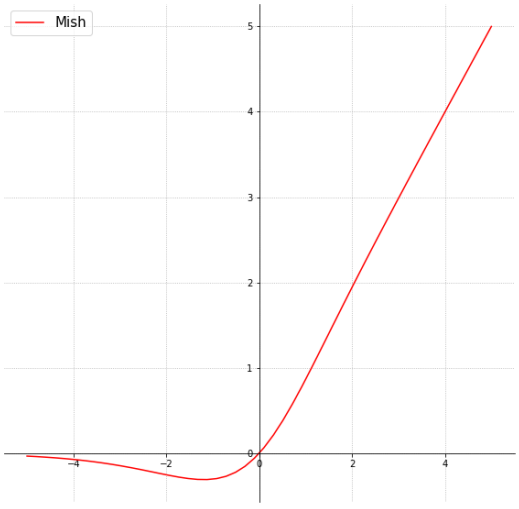


Slika 3.10: Mosaic data augmentation example [8].

To deal with the object occlusion, YOLOv4 modifies the CutOut technique, which sets pixels of a random region to random value, in a way that one part of the first image is pasted onto the second, augmented one. The class label is modified proportionally to the cropped image size. A very nice side effect that this method gives is that it effectively reduces batch size without sacrificing performance. MixUp can also be considered a data augmentation technique that chooses two coefficients, superimposes two images and adjusts labels according to saved coefficients. DropBlock, DropConnect and DropOut are CutOut equivalents that operate on feature maps. All three methods are quite similar but DropBlock is considered more useful in convolutional networks because features of one whole contiguous block are discarded. DropOut is usually applied on fully-connected layers.

Interestingly, YOLOv4 uses evolutionary algorithms to derive the optimal set of hyperparameters.

YOLOv4 performs regularization not only on the image and feature map level but also on a final class vector. For example, in a task where four different classes exist, the ground true class vector could be $[0, 0, 0, 1]$. The label smoothing process modifies this in a sense network doesn't push necessarily towards the value 1 but rather towards some smaller value to reduce the model's chance to overfit. After applying label smoothing the vector could become $[0, 0, 0, 0.9]$.



Slika 3.11: Mish activation function [11].

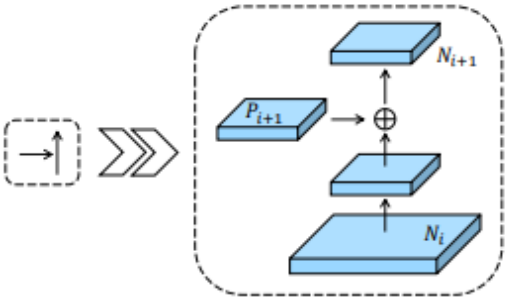
A bag of specials improve the model's accuracy without worsening the model's inference performance. In terms of YOLOv4, postprocessing boundary boxes, adding an attention mechanism and using the mish activation function are some of the most im-

portant methods. The mish activation function is a non-monotonic and self-regularized that works great when combined with data-augmentation techniques. Mathematically, it is defined as:

$$\begin{aligned} softplus(x) &= ln(1 + e^x) \\ f(x) &= x * tanh(softplus(x)) \end{aligned} \tag{3.14}$$

3.4.5. YOLOv5

YOLOv5 is a model that led to a lot of controversies because no paper was published officially. The model uses Cross Stage Partial Networks(CSP) as a backbone network which introduces several novelties for obtaining richer gradients while optimizing calculation cost for $\approx 20\%$. The core of such network is splitting gradient flow into multiple branches by first splitting the base layer's feature map into two parts and then merging those two parts through the cross-staged hierarchy. With its architecture, CSP networks manage to distribute the computation evenly across all layers. Hence, arithmetic units are optimally used and the probability of bottlenecks is significantly reduced. Cross-channel pooling is used to compress feature maps and consequently, memory cost can be reduced up to 75% on PeleeNet networks.
This model uses a concept of feature pyramids to help models generalize better on objects occurring at different scales and sizes. Feature pyramids, in YOLOv5, are part of the Path Aggregation Network for Instance Segmentation(PAN). All feature pyramid networks, including PANs, are composed of two pathways, bottom-up and top-down. In bottom-up pathways, a large number of convolutional layers is used to decrease the spatial resolution and to obtain semantically more meaningful feature maps.



Slika 3.12: Building block used in the bottom-up pathway in path aggregation [9].

In the top-down pathway, higher resolution layers are getting reconstructed and connected with corresponding feature maps using lateral connections to improve the

detections' precision.
The last part of YOLOv5, called the model's head, is the same as in YOLOv3 and YOLOv4. It is responsible for generating bounding boxes, confidence and class probabilities.
Contrarily to its predecessor, YOLOv5 uses the Leaky ReLU activation function in all layers except the last one, in which the sigmoid function is used as the activation. For gradient calculation, developers can choose between binary cross-entropy and focal loss and there are also two optimizers supported: Stochastic Gradient Descent and ADAM.
YOLOv5 improves on the k-means algorithm for choosing prior bounding box sizes by extending it with the evolutionary algorithm.

Algoritam 2 Evolutionary algorithm for choosing prior bounding boxes.

```
Input
D = dataset
Output:
List[BoundingBox] = list of bounding boxes
Main:
initial_bboxes = KMeans(9, D)                          ▷ K-means clustering
while can be further optimized do
    bbox_1, bbox_2 = GetTwoRandomBoundingBoxes(initial_bboxes)
    crossed_bbox = Crossover(bbox_1, bbox_2)
    mutated_box = Mutate(crossed_box)
    ReplaceWithTheWorstBoundingBox(mutated_box, initial_bboxes)
end while=0
```

The model uses torch's distributed framework so that training on multiple GPUs could be possible both on a single machine as well as multiple machines. In such a setup, training turns into a single-program multiple-data (SPDMD) process in which the original dataset is split across multiple replicas and the framework takes care of correctly synchronizing gradient computation.
Same rules apply in the case of YOLOv5 as for other parallel applications. The user should always strive to minimizw the total amount of data being transferred between the GPU device and the CPU host. However, during transfer, it is best to send as much possible data so that per-transfer cost could be held on bare minimum. One of the most effective ways to reduce the communication cost is by interleaving it with a computation that can, when working with GPUs, be done with multiple kernels execution.

Almost all modern computers base their memory management strategy on the concept of virtual memory. Such a concept allows the program to work with entities called pages which use the page table to find its mapping place in the secondary storage. In this way, the program utilizes memory more efficiently because it works with a consecutive block of logical memory. However, it is impossible to transfer the data directly from the CPU to the CUDA because the GPU device works only with page-locked (unpageable) memory. To solve this, torch introduced a concept of pinnable memory which is page-locked and needs to be reserved before the communication fires away on the host side. This incurs a significant communication cost because one chunk of data needs to be sent twice, first on the host side from the paged memory to the page-locked data buffer and second time from the data buffer to the GPU device. However, by pinning data buffers using the torch framework, the cost reduces when multiple communication steps need to happen.

4. Object tracking in computer vision

4.1. Introduction to object tracking

Detecting objects is a useful feature for computer vision systems but very often one would like to track objects through a sequence of frames to analyze trajectories. Object tracking extends object detection algorithms in a way that detected objects are assigned unique IDs and tracked through time. The simplest algorithm one could use to implement object tracking is to consistently apply an object detection algorithm in each frame. In a tracking set-up, except for the existence of the temporal dimension, objects can be identified even in frames when the detector fails to detect them because of advanced methods from estimation theory that can approximate an object's location from previous ones. This feature is extremely important since systems based on object tracking are deployed at many complicated places in various industries and many things could influence temporal detector failure.

There are many possible ways to divide object tracking and the first one that described is based on its application. In video tracking, estimation theory is extremely useful since it provides us with a framework for combining measurements with a prediction provided by a system model. Image tracking found its largest application in Augmented Reality scenarios where customers can see how an item they bought could look in their home.

In another split, single-object tracking and multiple-object tracking can be defined. As its name says, single object tracking is used for tracking the single object, which is detected in the frame, throughout the video. Its biggest advantage is efficiency since no detection algorithms are necessary. In multiple object tracking, objects can be of any class and are detected in each frame. MOT can be further divided into separate trackers and joint trackers. Separate trackers employ the strategy of tracking-by-detection in which target localization happens in every frame. Join trackers jointly train detection, embedding, motion and association models and their biggest advantage is low computational cost.

Based on the environment in which they are used, there are offline and online object-tracking algorithms. Offline trackers learn objects' features while having whole video footage at their disposal. On the other hand, online trackers learn features in parallel as they do predictions. Online and offline expressions are also used to distinguish trackers that work on real-time videos and recorded ones.

Four steps can be identified in every object-tracking pipeline. The process starts with target initialization in which bounding boxes are drawn around objects in the first frame. In appearance modelling, target objects are being modelled so that model could improve its robustness due to effects like motion, rotation, lighting and different angles taken for recording. In visual appearance, the object's features are constructed while statistical appearance uses advanced mathematical models for effective object identification. Motion estimation predicts an object's next position using mathematical models while in target positioning, input from a visual model is used for a more precise inference of the object's exact location.

There are two biggest problems object trackers are dealing with. Missing association occurs when one object was assigned multiple IDs during the video's life span so there are more than one tracks. Most often, this problem is addressed at the end of the tracking process by connecting tracks that belong to the same object. Such methods rely on spatiotemporal information obtained during the tracking as well as appearance information obtained as object features. Missing detection occurs when an object is not detected but is rather classified as the background. This manifests mostly in low-resolution and low-quality videos with a lot of noise and possible occlusion situations. Solutions for missing detection problems are mostly based on linear interpolation or, as we will revisit later, some kind of detection noise filter.

4.2. Object tracking performance metrics

Although usually implemented by iteratively applying object detection model, object trackers's performance can be evaluated and expressed independently from the detector. Multiple object tracking accuracy (MOTA) combines three sources of errors: number of false positive examples, number of false negative examples and number of identity switches in the following manner:

$$MOTA = 1 - \frac{\sum_k FN_k + FP_k + IDSW_k}{\sum_t GT_k} \tag{4.1}$$

where k denotes current frame and GT is the number of ground true objects in

the current frame. Although very expressive, MOTA is rarely used alone as the performance indicator. Multiple object precision (MOTP) measures tracker's localization ability by averaging overlap between ground true and bounding box detections over all frames. It can be expressed as:

$$MOTP = \frac{\sum_k \sum_i d_{k,i}}{\sum_k c_k} \tag{4.2}$$

where k denotes the current frame and i is the object of interest. As object tracking deals with trajectories, one would like to measure quality of trajectories. Number of identity switches tells us how many times the ground true object was assigned with a new id. One should be careful when interpreting this measure because although smaller number of identity switches is desirable, the tracker that produces more trajectories will certainly result in a larger number of identity switches. Ground true trajectories can be classified as mostly lost (ML), partially tracked (PT) and mostly tracked (MT). The track is considered mostly lost when it was tracked for less than 20% of its life and mostly tracked if it was tracked for more than 80% of its life. In all other situations, the track is considered partially tracked (PT). Number of times that the ground trajectory changed its status from tracked to untracked is expressed with the number of fragmentations (FM).

4.3. SORT object tracking algorithm

The SORT family of object tracking algorithms is a representative of the tracking-by-detection approach where all objects are detected in every frame. The first SORT model is intended to be used in real-time applications. It completely ignores appearance features and uses only mathematical algorithms for motion estimation and data association.

A Recursive Kalman filter is used for approximating positions in each frame for all detections in the previous scene with the assumption of using a constant velocity model. The SORT model uses a 7-dimensional vector to represent the state of every detection in some frame k:

$$x_k = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}] \tag{4.3}$$

where u and v are detection's centre coordinates, s represent bounding box's area and r is the detection's aspect ratio. Centre coordinates are assumed to follow the constant velocity model as well as the bounding box's area and are denoted by $\dot{u}, \dot{v}, \dot{s}$.

Kalman's filter has two phases, state prediction and state update. In the state prediction phase, the new state is calculated in the following way:

$$\begin{aligned} \hat{x}'_k &= F_k \hat{x}_{k-1} \\ \hat{P}'_k &= F_k P_{k-1} F_k^T + Q_k \end{aligned} \tag{4.4}$$

where x_{k-1} and P_{k-1} represent mean and covariance at the frame $k-1$, F_k is the state transition matrix, Q_k is the process noise covariance matrix and $\hat{x}'_k$ and $\hat{P}'_k$ are estimated mean and covariance at the current frame k. The state update step starts with calculating Kalman gain K as:

$$K = P'_k H_k^T (H_k P'_k H_k^T + R_k)^{-1} \tag{4.5}$$

where R_k is the observation noise and H_k^T represents observation model. The Kalman gain is then used in obtaining the final estimates. Using z_k as the measurement of any state variable in the current frame, the final estimate is obtained using the:

$$\begin{aligned} x_k &= \hat{x}'_k + K(z_k - H_k \hat{x}'_k) \\ P_k &= (I - K H_k) P'_k \end{aligned} \tag{4.6}$$

After running the Kalman filter, there are two sets of objects of interest. From the previous frame we have a piece of information about all existing detections, and the current frame gives us estimates for new bounding box positions.
The SORT algorithm uses the Hungarian algorithm to solve the minimum cost assignment problem. In its most general form, the algorithm solves a problem of minimizing the cost of assigning n workers to n jobs such that each assignment has some cost, each job is solved by exactly one worker and each worker solves exactly one job. This can be modelled as a bipartite-matching problem using graph theory but such a definition will be avoided here and instead, a mathematical definition will be provided:

$$\arg\min_x \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{4.7}$$

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1 \\ \sum_{j=1}^{n} x_{ij} &= 1 \\ x_{ij} &\in \{0, 1\} \end{aligned} \tag{4.8}$$

The brute-force solution runs in $O(n!)$ complexity but using the Hungarian algorithm it can be solved in polynomial time, $O(n^3)$.
In the context of object tracking, workers are objects from the previous frame and jobs are detections in the current frame so the problem can be formulated as matching exactly one existing object to the new detection. Intersection over Union is used as the association cost metric but instead of minimizing, one would like to maximize the algorithm's cost.

4.4. DeepSORT object tracking algorithm

SORT proved that is possible to construct simple and effective object-tracking solution especially useful for real-time applications. Its biggest problem is a large number of identity switches because of failing to track the object for a longer time when occlusion or some other malign effect occurs. DeepSORT tried to address this by integrating appearance information into the tracking pipeline and combining it with the motion information. For this, a convolutional neural network with the architecture shown in the image 4.1 has been pre-trained on a person re-identification dataset.

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Slika 4.1: Architecture of CNN used in DeepSORT for calculating appearance information [20].

Motion information is obtained in a similar way as in the original SORT model, by using the Kalman filter and Matching Cascade algorithm for solving a series of cost-assignment problems using the Hungarian algorithm. In the Kalman filter, the only change is related to the state which now becomes an 8-dimensional vector:

$$x_k = [u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h}] \tag{4.9}$$

with u and v representing 2D detection's centre coordinates, $\dot{u}$ and $\dot{v}$ their speed, γ aspect ratio. DeepSORT also assumes that aspect ratio has speed $\dot{\gamma}$ and instead of

working with scale (area), this model measures height h and its speed $\dot{h}$.
To quantify motion information, squared Mahalonobis distance is used between new detections and predicted states obtained as the output from the Kalman filter:

$$d^{(1)}(i,j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \tag{4.10}$$

This distance measure proved to be extremely useful for estimating objects' position frame-to-frame where the object is continuously seen. The appearance information requires defining the appearance descriptor for each bounding box and saving it for the last L_k frames. It can then be expressed as the cosine distance between the i-th track and j-th detection as follows:

$$d^{(2)}(i,j) = min\{1 - r_j^T r_k^{(i)} | r_k^{(i)} \in R_i\} \tag{4.11}$$

Taking into account cosine distance between existing tracks and new detections proved to be extremely useful in situations when the object is occluded for some time because it is possible to recover its identity. Two metrics are then combined:

$$c(i,j) = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{4.12}$$

to obtain a single measure of association cost between tracks and detections. However, DeepSORT is mostly used in a setup in which motion distance is used only for discarding outliers and the associated cost is equal to the appearance information. This corresponds to the case when $\lambda = 0$.

4.5. StrongSORT object tracking algorithm

StrongSORT can be considered as a set of improvements over the existing, DeepSORT model. Instead of the Faster R-CNN detector used in the DeepSORT, StrongSORT uses the YOLOX-X model extended with BoT for feature extraction. The basic Kalman filter, as used in the SORT and DeepSORT, suffers from low-quality detections and doesn't take into account the scale of the detection noise. That's why NSA Kalman is used which introduces one subtle but very important change to the original implementation:

$$\hat{R}_k = (1 - c_k) R_k \tag{4.13}$$

In this formula, R_k is the covariance matrix of measurement noise and c_k is the detection confidence in frame k. In this way, when detection confidence is high, the

measurement noise can be reduced and the algorithm can weigh more the state update step than it would otherwise. Contrarily to DeepSORT, StrongSORT calculates association cost from both, motion and appearance information with $\lambda = 0.98$ set in equation 4.12. This model also uses a simpler assignment scheme than in DeepSORT to reach better performance.

StrongSORT++ uses StrongSORT as the baseline model and enhances it with AFLink and GSI to deal with the missing association and missing detection. As opposed to some other highly-performant trackers, AFLink tries to connect tracks using only spatiotemporal information. It consists of two simple, non-weight-sharing convolutional neural networks, named temporal and fusion modules each producing its own feature set. Track T_i can be defined as follows:

$$T_i = \{f_k^i, x_k^i, y_k^i\}_{k=k^i}^{k^i+N-1} \tag{4.14}$$

As the last step, feature sets of two tracks are concatenated and passed through a simple multi-layer perceptron(MLP) to predict the association score.

GSI uses Gaussian processes as an alternative to linear interpolation for dealing with missing detection. Let's say that at the frame k, the object i cannot be observed due to some random effect, e.g motion or blur. Object's new position p_k can still be estimated by using the Gaussian process:

$$\begin{aligned} p_k &= f^i(k) + \epsilon \\ f^i(k) &\in GaussianProcess(0, r(p_k, p_{k-1})) \\ r(p_k, p_{k-1}) &= exp(-\frac{||p_k - p_{k-1}||^2}{2\lambda^2}) \end{aligned} \tag{4.15}$$

where r is the radial basic kernel function, λ is a hyperparameter controlling the trajectory's smoothness and $\epsilon \sim N(0, \sigma^2)$ is a Gaussian noise.

4.6. Other object tracking algorithms

MDNet is a single-object tracker that is most often used in an offline setting. During the training step, videos from several domains are used to better learn the object's features. These videos are encoded in multiple fully connected layers and removed during the model's work in an online mode to reach high performance.

When introduced, GOTURN was the first model that was trained in an offline setting to speed up the online usage mode. Same as MDNet, it is also a single-object tracker

that is learned on thousands of videos in its training step. The model forms prediction as a regression problem that takes as input the previous and the current frame.
Recurrent YOLO (ROLO) uses a combination of convolutional and Long Short Term Memory (LSTM) neural networks. LSTM uses features extracted by convolutional layers and predicts the bounding box coordinates of the object being tracked. ROLO is considered one of the most stable models due to its predictions even in situations when the object is occluded.

5. Development of the player tracking system

6. Zaključak

Zaključak.

LITERATURA

[1] Arthur Francisco Araujo Fernandes, Joao Dorea, i Guilherme Rosa. Image analysis and computer vision applications in animal sciences: An overview. *Frontiers in Veterinary Science*, 7:551269, 10 2020. doi: 10.3389/fvets.2020.551269.

[2] Alexey Bochkovskiy, Chien-Yao Wang, i Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL https://arxiv.org/abs/2004.10934.

[3] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Jonathan Hui. Mean average precision in object detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173, 3 2018.

[5] Jeremy Jordan. An overview of object detection: one-stage methods. https://www.jeremyjordan.me/object-detection-one-stage/, 2018.

[6] Amro Kamal. Yolo, yolov2 and yolov3. https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899, 2019.

[7] Ayoosh Katuria. Intro to optimization in deep learning: Gradient descent. https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent, 2018.

[8] Thananop Kobchaisawat, Thanarat Chalidabhongse, i Shin'ichi Satoh. Scene text detection with polygon offsetting and border augmentation. *Electronics*, 9:117, 01 2020. doi: 10.3390/electronics9010117.

[9] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, i Jiaya Jia. Path aggregation network for instance segmentation, 2018. URL https://arxiv.org/abs/1803.01534.

[10] Manolis Loukadakis, José Cano, i Michael O'Boyle. Accelerating deep neural networks on low power heterogeneous architectures. 01 2018.

[11] Tariq Rahim, Syed Hassan, i Soo Shin. A deep convolutional neural network for the detection of polyps in colonoscopy images. 08 2020.

[12] Joseph Redmon i Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, i Ali Farhadi. You only look once: Unified, real-time object detection, 2015. URL https://arxiv.org/abs/1506.02640.

[14] Simplilearn. What is computer vision: Applications, benefits and how to learn it, 2023. URL https://www.simplilearn.com/computer-vision-article.

[15] StatsPerform. Optavision. https://www.statsperform.com/opta-vision/, 2023.

[16] Luc De Raedt Tinne De Laet. Markov networks, 2021. URL https://onderwijsaanbod.kuleuven.be/syllabi/e/H02D2AE.htm#activetab=doelstellingen_idp1444992.

[17] Tzen Szen Toh, Frank Dondelinger, i Dennis Wang. Looking beyond the hype: Applied ai and machine learning in translational medicine. *EBioMedicine*, 47, 08 2019. doi: 10.1016/j.ebiom.2019.08.027.

[18] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, i A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013. doi: 10.1007/s11263-013-0620-5. URL http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[19] Web. Inverse filtering.

[20] Nicolai Wojke, Alex Bewley, i Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017. URL http://arxiv.org/abs/1703.07402.

[21] Fatemeh Zabihollahy. *Deep Learning Methods for Abnormality Detection and Segmentation in Computed Tomography and Magnetic Resonance Images*. Doktorska disertacija, 08 2020.

Naslov

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.