BRAIN. Broad Research in Artificial Intelligence and Neuroscience
e-ISSN: 2067-3957 | p-ISSN: 2068-0473
Covered in: Web of Science (ESCI); EBSCO; JERIH PLUS (hkdir.no); IndexCopernicus; Google Scholar; SHERPA/RoMEO; ArticleReach Direct; WorldCat; CrossRef; Peeref; Bridge of Knowledge (mostwiedzy.pl); abcdindex.com; Editage; Ingenta Connect Publication; OALib; scite.ai; Scholar9; Scientific and Technical Information Portal; FID Move; ADVANCED SCIENCES INDEX (European Science Evaluation Center, neredataltics.org); ivySCI; exaly.com; Journal Selector Tool (letpub.com); Citefactor.org; fatcat!; ZDB catalogue; Catalogue SUDOC (abes.fr); OpenAlex; Wikidata; The ISSN Portal; Socolar; KVK-Volltitel (kit.edu)
2025, Volume 16, Issue 2, pages: 418-434.
Submitted: March 4th, 2025 | *Accepted for publication:* May 11th, 2025

Region of Interest Detection Using AI Techniques and Morphological Operations

Andrei Gabriel Nascu
University of Craiova, Craiova, Romania.
andrei.nascu@edu.ucv.ro
https://orcid.org/0000-0003-2608-5463

Abstract: *Nowadays, with the increase of technological advancement, AI can be found anywhere, from smartphones with facial recognition, voice assistant, and text autocorrect, to security systems to detect unauthorised persons, cyberbullying, cyberattacks, or even in healthcare to assist doctors in giving diagnostics and treatments. The intersection point of all these use cases is that they all need a lot of labelled data to obtain good results in classification processes. Labelling is tedious work that can consume many resources, like time, workforce, computer power, and money. The labelling process differs according to the type of data that you are labelling (audio, video, images, text). This paper focuses on the image type of data. To solve this issue, the paper proposes the use of an auto-labelling algorithm by employing a combination of a Convolutional Neural Network (CNN) architecture and morphological operations with a set of hyperparameters. To quantify how well the algorithm performs, an efficiency metric given by two components, a local evaluation of the results and a global evaluation of the results, is established by comparing the labels given by the algorithm with the labels manually done. Furthermore, the paper analyses the impact of the hyperparameters on the algorithm using the established efficiency metric. Finally, the results for the best configuration of the hyperparameter are explored using both global and local data analysis. Using the efficiency metric given by the overlap area, the algorithm achieved a mean overlap percentage of 89.33%.*

Keywords: *labelling; region of interest; CNN; morphological operations; image masks.*

How to cite: Nascu, A. G. (2025). Region of interest detection using AI techniques and morphological operations. *BRAIN: Broad Research in Artificial Intelligence and Neuroscience*, *16*(2), 418-434.
https://doi.org/10.70594/brain/16.2/30

1. Introduction

The Artificial Intelligence (AI) domain has increased drastically in the last decades to the point where in today's society, a large part of our lives is guided by a form of it. As the industrial revolution and the digital era have brought lots of changes to everyday life, so will the AI boom that we are currently experiencing (Makridakis, 2017). These developments are especially interesting to track in the job market, where AI is making substantial changes. At the beginning of this new era, such transformations are outlined by Neufeind, O'Reilly, & Ranft (2018), where it is stated that in most developed countries, AI determines the following:

- less demand for routine jobs, the so-called "middle jobs" like clerks and machine operators;
- more demand for non-routine jobs, like researchers, doctors, and engineers, who all require a high level of education;
- the emergence and development of new jobs/activities that require special skills related to the AI domain.

Nowadays, AI models can be applied with notable performance outcomes in multiple fields like healthcare, transportation, agriculture, security, space exploration, and many other domains, each with its particular use case, database, and purpose (Marr, 2023). Each of these branches can use one or more models that can analyse images, texts, sounds, actions, or even simple numbers. The common thing for all these models, for achieving high-performance scores, is that they need a lot of data. To give an example of the importance of big data in creating AI models, a study by Sun et al. (2017) was conducted, in which a vision model was trained and evaluated on different dataset scales. The model's performance increased logarithmically with the volume of data that was used. In visual tasks, many convolutional neural network (CNN) architectures have the feature extractor component pre-trained on a large dataset like ImageNet, also named ILSVRC. One of the most commonly used branches is ILSVRC 2012, which has 1000 classes and over 1.2 million images for training (Hugging Face, n. d.). For any supervised learning algorithm, all this data must be labelled, which might become an extremely long and tedious process.

The labelling process represents the action of identifying which object or class is in the data (image, video, sound, etc.) and indicating where this class is in the analysed data (Amazon, n.d.). The action of labelling is one of the emerging activities that are needed in the new AI-driven world that depends on the data. In some cases, it may require the labeller to have deep knowledge of the subject, for example, in the healthcare field. To obtain good labelling of data, it can be used tools like Label Studio, LabelMe, LabelBox, etc.

To further emphasise the importance of the labelling process in visual tasks, especially for autonomous vehicles, a few examples of training models in recent papers are presented below:

- Khan, Lee, & Lim (2023) trained a hybrid model that unites two well-known architectures, namely YOLO and Fast R-CNN, on 10000 labelled images from traffic scenarios;
- Jia et al. (2023) improved the YOLOv5 model to detect small objects better and obtain better accuracy by employing structural re-parameterisation; the model was trained on the KITTI dataset, which has 15000 images;
- Rahman, Azad, & Hasan (2022) proposed a solution for detecting small objects in a heavily loaded image; the new model's accuracy was measured using the Dhaka AI dataset, which has approximately 4000 images.

In this paper, we propose an algorithm that tries to alleviate the problem of labelling from humans. Compared to other methods, the algorithm combines two key aspects: the usage of a pre-trained CNN and the flexibility of using morphological operations to isolate different foreground objects. For this, the algorithm uses image processing techniques like threshold, morphological operations on images (erosion, dilation), colour conversion, contour detection, background removal, etc. The key aspects of the paper are listed below:

- obtain the labels of the main frontal object for each image
- construction of the database of images from a movie

- compute an evaluation technique to analyse the correctness of the obtained results.

The rest of this article is structured as follows. Firstly, it has a description of the evolution of AI with one of the main activities associated with it, namely the labelling process. A short analysis of the articles that are on the same topic of labelling images is presented. Then, a new solution offered for the labelling problem is described. Subsequently, the article focuses on the process of data acquisition from a video file, the hypotheses of the experiment, and the performance of the algorithm. Lastly, the key aspects of the paper are highlighted, and some potential developments of future research are proposed.

2. Related Works

An improvement to the Auto-context algorithm to obtain better results in constructing feature maps was proposed by Ji, He, & Yang (2014). The authors use an iterative procedure to approximate a change in the scale of the foreground object. After obtaining the scale, it can determine the best radius of the neighbourhood of a patch that is used to train the classifier. Experiments have shown an improvement of the initial algorithm in both the F-measure and precision-recall curves.

Another work in the labelling field was done by Fenchel, Thesen & Schilling (2008), where they constructed an atlas of medical images of the torso, which has two main components, a model for the deformation and a model for grey value appearance. To improve the segmentation/labelling, an algorithm is used to eliminate the arms from the images that compose the atlas. The alignments of the images in the atlas are done using both affine transformations and free-form transformations, resulting in deformations on the initial images. The atlas also contains manual landmarks added for each of the anatomical structures. The labels for each of the anatomical structures are propagated to the new image from the atlas, after aligning the new image (affine transformations and deformations) to the images of the atlas.

A solution for autolabelling of images is given by Zhang et al. (2020), where they associate categories for subsets of images and tags for each image. The authors propose an algorithm that combines two well-known methods for extracting feature vectors from images, namely Spatial Pyramid Matching (SPM) and Convolutional Neural Networks (CNN). The two vectors are then combined to give a metric that can calculate the distance between two images, one from the test set and the other from the training set, with the importance of each vector given by a hyperparameter alpha for SPM and 1-alpha for CNN. To reduce the computational complexity, the authors suggested using noise filtering that calculates the category class for the test image, and then it will only analyse the images from the training set that are in the same category as the test image. After calculating the distances to all the images from the same category, the tags from the most relevant/similar images are propagated to the test image. This method has a downside that it isn't able to give spatial information about the tags from the images.

Labelling images of DLOs like wires, cables, and tabulator structures obtained by using a robotic arm with a camera is described by Caporali et al. (2023). Their approach is based on human involvement by using a sensor tracked in space, namely a VR pen to draw lines. Using homogenous transformations, the coordinates given by the sensor to the base station are then transformed into camera coordinates, and so the labels are obtained. The labelling process is further enhanced to eliminate human errors by using a CNN architecture to calculate offsets for the label points previously produced.

A two-component AI architecture composed of an AdaBoost classifier and a Bayesian Network (BN) model to detect facial expressions like sadness, happiness, boredom, and disgust, by analysing facial actions of the nose, lips, brows, and lids, is proposed by Zhang, Tong & Ji (2008). The BN is modelled as a graph that captures the relationships between the facial actions, namely co-occurrences or exclusions. This model uses an iterative process guided by limited human interaction to enhance the predictions. The AdaBoost gives a decision for each of the facial actions, which are then ordered by information gain, so that a minimal amount of changes given by the user would determine big information gained for the Bayesian Network. For each of the

facial actions, the BN model will integrate at each iteration two components: the AdaBoost automatic decision and the new information obtained from the human interaction, with weights given by the accuracy of the AdaBoost classifier and, respectively, by the knowledge of the human labeller.

An improvement of an existing method, namely upgrading from the SmartLabel to a new version of SmartLabel2, is described by Wu & Yang (2009). The initial version splits an image into a grid system of patches. Then the method needs a human input of a small rectangle inside the region of interest (ROI) that the user wants to detect. All the patches will be arranged in a graph structure with values between 0 and 1 for each vertex (patch), after the labelling. The graph's edges are computed based on a similarity function between adjacent vertices. Initially, the method has only positive values of patches given by the drawn rectangle. A patch with a value of 1 means it is complete in the rectangle area, and a value of 0 means the respective patch is completely outside the ROI area. The values between 0 and 1 give a partial overlap between the patch and the border regions of the rectangle. This method is iterative to find patches that are not a part of the detected object, and it requires further human feedback. The upgraded version still requires a drawn rectangle for the initial step. Still, it adds further improvements like autodetection for the patches that are not part of the object, implementing a new method of drawing patches using quadtrees, and the use of Normalised Cuts to partition the analysed image into superpixels.

A multiscale conditional random field is proposed by He, Zemel, & Carreira-Perpinan (2004), where a model with three different levels of granularity is created to solve the labelling for each patch of a set of images. The main components of the model are: a local classifier implemented by a neural network, like a multilayer perceptron that acts on a small number of pixels, centered on a certain pixel I, which is the labelled one, regional label features, that act on a larger scale by detecting relationships between different objects, and a global label features implemented as a restricted Boltzmann machine, that analyses the image as a whole. The purpose of combining these components is to obtain a complementary functioning, where one of the components does not perform well, and the other two correct the decision. This behaviour is especially important for areas that could not be distinguished at a local level, like the sky and water, which are both blue. To approximate the parameters of the model, the authors use a contrastive divergence algorithm. The training process uses a dataset that is split into two subsets of images for training and testing, with the three components trained sequentially.

Unlike the methods that have been presented so far, the main direction of this article is given by a few points, like the reduction of human intervention, the absence of a training dataset, the usage of morphological operations, and the flexibility offered by a set of hyperparameters. After employing a CNN architecture that eliminates the background, the algorithm identifies the main object from the foreground, the one with the widest area. A series of morphological operations is applied on the mask until a certain convergence criterion is reached, thus giving the algorithm an iterative nature.

3. System Design and Implementation

3.1. Algorithm's Functional Description, Input, and Output

The proposed algorithm identifies the coordinates of the bounding box that best suits the main object in the image. It also may create, depending on the desired debugging and tracking selected options, a series of subfolders in the main folder named database, these folders being the following:

- **Foreground:** where the initial images with the main object are removed and the background is saved. (optional)
- **Grey:** where the images from the foreground are converted to greyscale. (optional)
- **Labels:** where the coordinates of the bounding box for the main object are saved, each image has a corresponding text file with the same name.

- **Masks:** where binary images (black for the background and white for the object/foreground) are saved after applying morphological operations on the grey images. (optional)
- **ROI:** where the initial images with the bounding box drawn on them are saved. This is the final output of the algorithm, and its purpose is to be able to visually evaluate the results for the detection. (optional)

Firstly, the algorithm loads each image from the Images folder using the OpenCV library, as can be seen in Figure 1 (left). Then, it uses a neural network architecture embedded in the *rembg* library to be able to distinguish between the foreground objects and the background in each image (Das, 2023), thus obtaining a new image of the same size as the initial image, but only with the objects from the foreground as it can be seen in Figure 1 (Middle). Using only this approach, the problems that could arise are the following: there could be multiple objects in the foreground, and we are interested only in the most important object, the one that is always visible in the center of the video and also has the biggest area. The main object could be near another foreground object, a common case is when the object sits on a table or a box.

Figure 1. Left: the initial image. Middle: the image after removing the background. Right: the image converted to greyscale

To solve these possible problems, the previously obtained foreground images are first converted to greyscale as in Figure 1 (right). Then, considering that there is a colour difference between the main object and the object on which it sits, there will be applied a thresholding operation, with parameters that can be provided by the user. The newly acquired image, which can be viewed as a mask for the initial image, is then passed through a series of morphological operations to dilate the white area. Finally, we determine the largest contour for a white area in the image and we calculate the rectangle that best fits the contour. The coordinates are then normalised and saved in a text file using the same name as the image file.

Algorithm 1. Pseudocode for determining the region of interest in the image

```
MainPercentArea = 80; ThresholdValue = 80
ThresholdType = 0; KernelSize = 5
for each img in images do:
        foregroundImg = removeBackground(image)
        grayImg = convertColour(foregroundImg, GRAY)
        mask1 = threshold(grayImg, ThresholdType, ThresholdValue)
        mask2 = threshold(grayImg, ThresholdType, 1)
        contours = findContours(mask2)
        largestContour = max(contours)
        mask3 = drawContours(largestObjMask, largestContour, FILLED)
        mask = bitwise_and(mask1,mask3)
        do:
                mask = dilate(mask,kernel)
                regions, counts = countRegionsPixels(mask)
                totalWhiteArea = sum(counts)
                largestWhiteArea = max(counts)
                currentMainPercentArea = largestWhiteArea/totalWhiteArea*100
        while currentMainPercentArea < MainPercentArea
        contours = findContours(mask)
        if contours is not empty then:
                contour = max(contours)
                x, y, w, h = boundingRectangle(contour)
                saveCoordinates(x,y,w,h)
        endif
endfor
```

3.2. Hyperparameters of the Algorithm

MainPercentArea represents the desired percent of the biggest object (main object) from the total white area of the current mask. This hyperparameter can theoretically take values between 0 and 100, but higher values are generally desired. To better understand the usefulness of this concept, in the context of the proposed algorithm, we will analyse the evolution of the area of the main object for one of the images. Initially, the first mask is obtained after applying a thresholding operation on the greyscale image, thus resulting in a two colour, black and white image. In this mask, as it can be seen in the first image from Figure 2, there are multiple white areas with the biggest area representing the main object. We will calculate the dimension of the biggest white area and then we will divide this value by the total number of white pixels, thus obtaining the percentage value of the main object for the current image as in Equation 1:

$$CurrentMainPercentArea = \frac{CurrentMainArea}{TotalMaskArea} * 100 \quad (1)$$

Figure 2. From left to right: the initial mask resulted after the thresholding operation, the mask after the first dilate operation, the mask after the second dilate operation, and the last mask resulted after the third dilate transformation

If the *CurrentMainPercentArea* is greater than or equal to *MainPercentArea,* then the algorithm stops, and this will be considered the final mask. Otherwise, we will enter a loop in which the current mask will be applied to a morphological operation of dilation, resulting in a new mask in which the white area of the main object will have a bigger value. This loop will end only when *CurrentMainPercentArea* becomes greater than or equal to *MainPercentArea*. As it is expected, a greater value for the *MainPercentArea* will require a greater number of iterations, until the condition is satisfied. In this case, we can follow the evolution of the main object in each mask in Figure 2. The total white area and the dimension of the main object for each mask can be seen in Table 1 below.

Table 1. Evolution of the mask from the dilation loop

Iteration Number	Number of pixels of the white area	Number of pixels of the biggest continuous white area	Percent of the biggest white area of the total area
0	11636	3180	27.33%
1	29802	13366	44.85%
2	40338	31173	77.28%
3	*49264*	*48639*	*98.73%*

The workflow of the MainPercentArea hyperparameter in the context of Algorithm 1 is described in Figure 3.

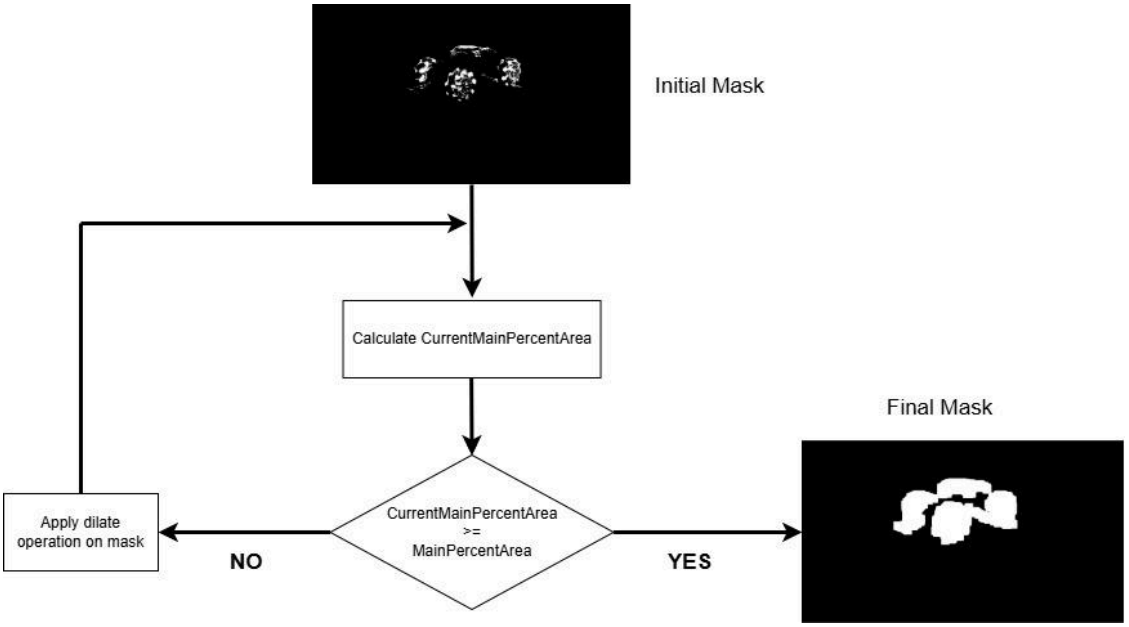

Figure 3. MainPercentArea hyperparameter in the dilate operation loop

ThresholdType is a binary value that indicates if the algorithm should use binary thresholding or inverse binary thresholding. Both of them divide an image into two classes corresponding to black and white. Taking a thresholding value of *x*, the first method will colour a pixel in white if it has a value above *x* and in black otherwise. The second method will have the opposite behaviour (Educative, n.d.). The first method (binary) should be used if the object that should be removed is a darker one and the one that must remain (the main one) has a lighter colour. The second method (inverse binary) should be used in the case of a lighter object that should be removed and a darker object that must be kept in the mask. To better understand the functionality of this hyperparameter, Figure 4 shows the effect of *ThresholdType* on a greyscale image with an arbitrary x value equal to 135. As it can be seen, the two images resulting from changing the value of the hyperparameter are complementary to one another.

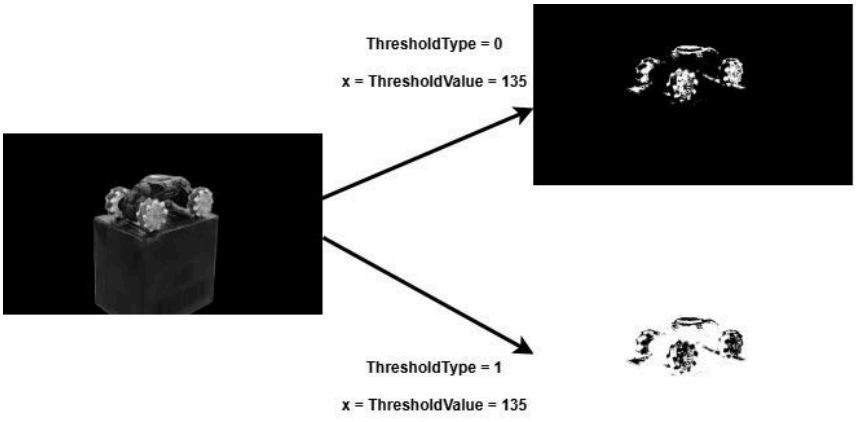

Figure 4. Effect of the ThresholdType hyperparameter change on an image

ThresholdValue is a value between 1 and 255 that is used to split the set of pixels of a greyscale image into two categories: black for the background and white for the foreground, thus resulting in a binary image (Bhargavi & Jyothi, 2014). Depending on the employed method, the white pixels will be the ones that are above the threshold value for binary thresholding, or they could be the ones that are under the threshold value for inverse binary thresholding. Figure 5 shows

the impact of changing this hyperparameter on an image with a ThresholdType=1. As expected, the size of the white area is inversely proportional to ThresholdValue.

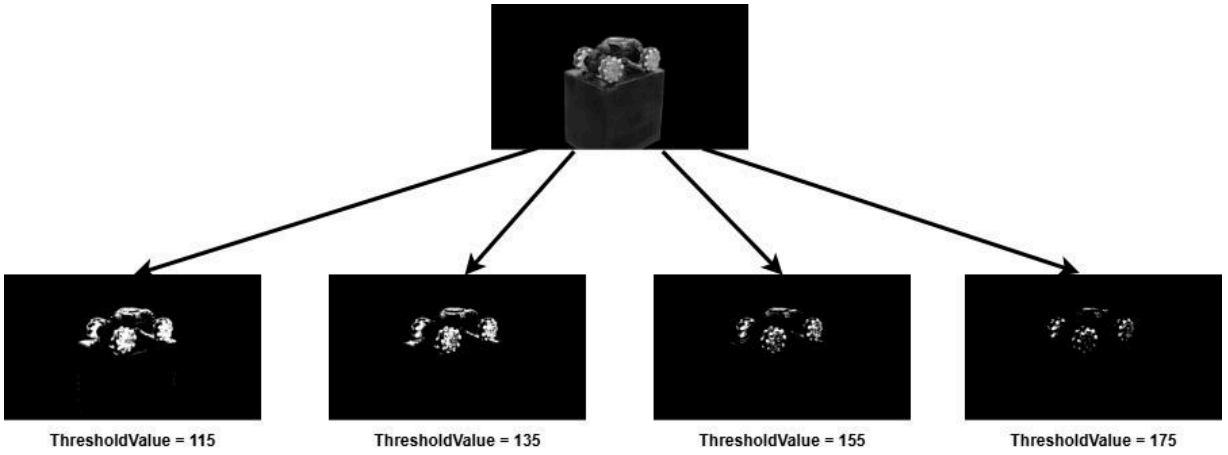

Figure 5. Image thresholding with different values for the ThresholdValue hyperparameter

KernelSize is used to create a kernel that will be one of the parameters for the dilation transformation. This morphological operation is used only on a binary image resulting from applying a thresholding operation (OpenCV, n.d.). The size of the kernel will dictate the increase of the white area of the image (the foreground) as it is directly proportional to the growth of the white area in the new image resulting after applying the dilation transformation. The dilation transformation's purpose is to unite non-contiguous white areas that could be parts of the same object, but were split by the thresholding operation. To better illustrate the functionality of this hyperparameter in the context of the dilate operation, Figure 6 presents a mask that uses different kernel sizes for the dilate operation.

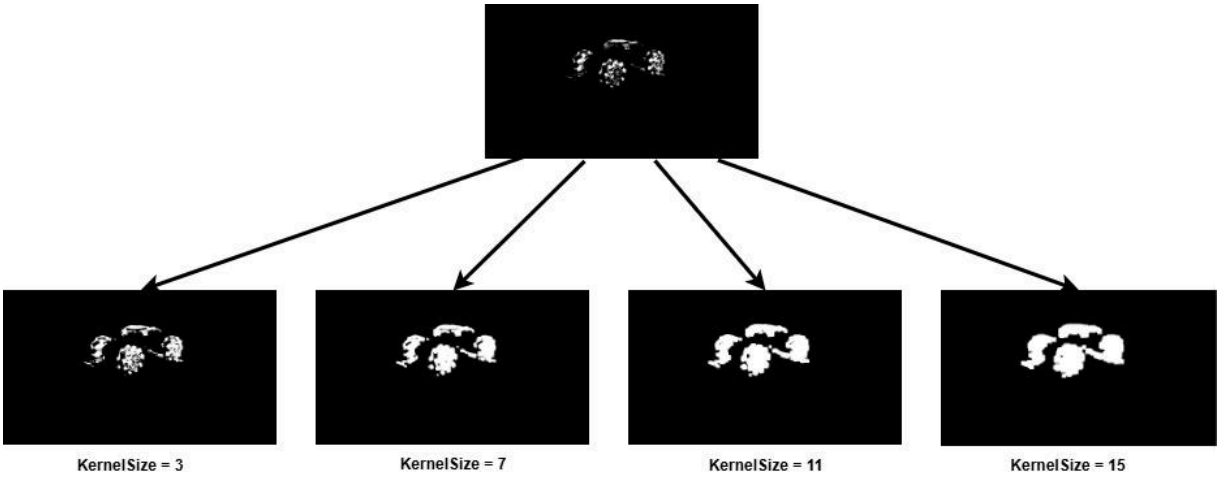

Figure 6. Impact of KernelSize hyperparameter on a mask

3.3. Artificial Intelligence Component for Background Removal

It is represented by the *rembg* library, which uses a neural network component to be able to separate an image into two main components: the foreground (object, person, building, or animal) and the background. This algorithm is an implementation by (Gatis, n.d.) of the U²-Net Model neural network architecture. The CNN architecture was developed by Qin et al. (2020) to perform visual segmentation tasks like salient object detection. Unlike many of the existing CNN architectures that are trained to classify images, the current model focuses on analysing local details and global contrast. This model is composed of two nested U-shaped structures that keep a high

resolution for the image without a big increase in either memory or computation costs. The U-structure has 2 main components:

- encoder, where there is a series of convolutional and pooling layers to downsample the original image;
- decoder, the opposite of the encoder, which has the purpose of reconstructing the original image by using upsampling operations.

4. Results

In this section, we analyse the results given by the algorithm using statistical metrics like mean and standard deviation. Furthermore, we will plot graphics that will highlight aspects like: number of iterations of the dilate transformation needed until the stopping criteria given by the hyperparameter *MainPercentArea* is met, data distribution in a histogram plot, comparison between current data distribution, and a normal/gaussian distribution.

The ground truth with which the results of the algorithm will be compared is the manual labelling of the image database. The manual labelling was done using the Label Studio tool. Both the algorithm and the manual labelling will give a result in the shape of a bounding box (rectangle), which is stored in text files in YOLO format. The performance criterion will be the percentage of overlap between the two rectangles, with higher percentages denoting better algorithm results. To further illustrate this metric in Figure 7, we have all the cases that could arise for two overlapped rectangles with the percentage of overlap calculated using Equation 2:

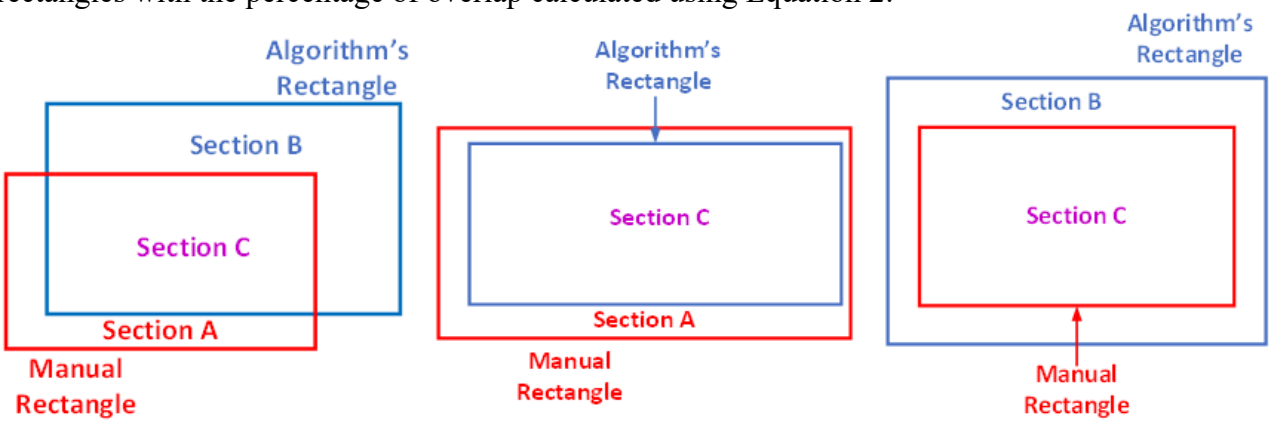

Figure 7. Left: Partial overlap between the result of the algorithm and the manual labelling. Middle: The algorithm detects only a portion of the ground truth given by the manual labelling. Right: The algorithm detects a wider area for the region of interest

$$OverlapPercent = \frac{Section\ C}{Section\ A + Section\ B + Section\ C} * 100 \quad (2)$$

This metric will be applied to each image of the dataset, resulting in a list of percentages, for which we will calculate all the statistics and plot all the graphics from above.

4.1. Database Building Process

In order to create the database of images, we are using a video file that is taken by a camera that revolves around the main object. The object of the study can be any type of thing, like a fruit, a car, a bicycle, an animal, a device, etc. Taking into consideration that for any object the algorithm of constructing the database would work in the same way, in the experiment we chose small objects like a toy car and an electric scooter. According to Haselhoff, Schauland, & Kummert (2008), one of the main characteristics of the videos that strongly influences most AI architectures that detect a region of interest corresponding to the main object is the image resolution. Considering that current models of the YOLO algorithm like version 11 (Khanam & Hussain, 2024), which is the current

state of the art, version 10 (Wang et al., 2024) or version 9 (Wang, Yeh, & Liao, 2024) use images of 640x640 pixels, we have chosen a resolution of 960x544 for the video.

The algorithm takes a video file and creates a new folder named database in which it will create a subfolder named images. After opening the video, the algorithm will extract and save each frame that is at a position multiple of *x*, with *x* being chosen by the user as a downsampling factor for the frames of the video. The advantages of using a value *x* different from 1 are the following:

- Eliminating similar images (frames that are one after another and do not add too much information).
- Creating a smaller database with the relevant images for a model to train, thus accelerating the training process.

Taking into consideration the *x* value, the frames per second (*FpS*), and the time of the video, the resulting number of images (*NrImg*) of the database will be given by Equation 3.

$$NrImg = \frac{time * FpS}{x} \tag{3}$$

In the case of one of my examples in which I had used a rate of 50 *FpS* with a video of 20.5 seconds and an x value of 3, the number of images resulted by using the formula from above is 341. In Figure 8, we present some of the resulting 341 images. In order to see the movement of the camera, we have chosen to show every 15 images from the dataset.

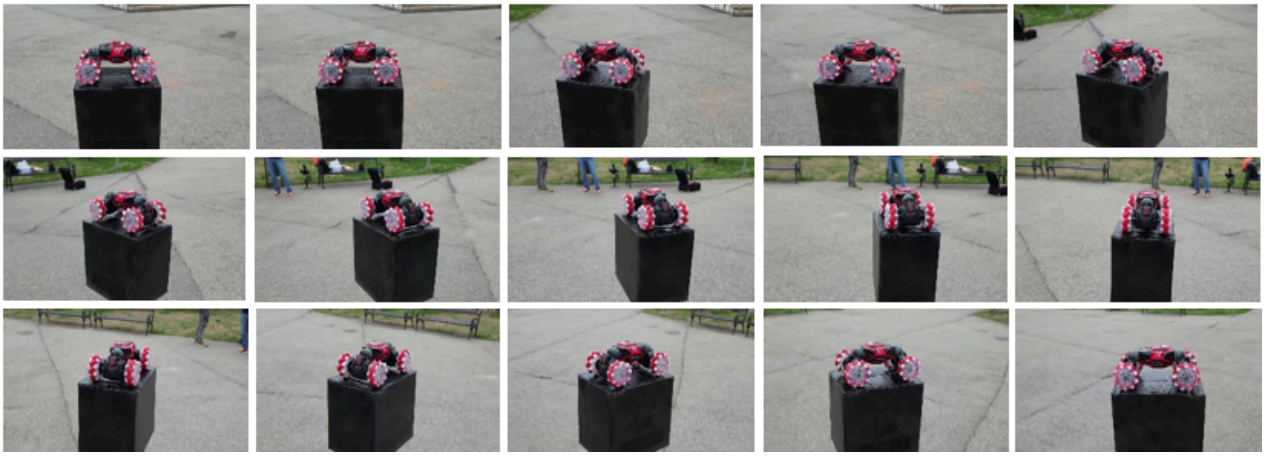

Figure 8. Images extracted from a video file

To better illustrate the workflow of this part of the program, the pseudocode of the proposed solution is shown in Algorithm 1.

Algorithm 2. Pseudocode for building the database

```
x=35
index=0
nrImg=0
videoFile=loadVideo(videoPath)
for each frame in videoFile do:
if index%x==0 then:
       imageName=concatenate(”car_”,nrImg,”.jpg”)
       imagePath=concatenate(“database/images/”,imageName)
       saveImage(imagePath,frame)
       nrImg+=1
endif
index+=1
endfor
```

4.2. Determine the Best Setup for Hyperparameters

In order to determine which configuration of the hyperparameters gives the best results, we have run the algorithm under the same conditions of input data, in this case, the same video but with different values for the hyperparameters. For KernelSize, the experiment started with a dimensionality of 5 and increased by 2 up to the value of 9, where it was observed that the best Mean Overlap Percent decreased to 87.69% from 89.33%, for a KernelSize of 5. For a dilate operation, the dimension of the kernel is recommended to be an odd number for two main reasons: firstly, to ensure that one of the pixels from the kernel is the center, and secondly, to apply the operation symmetrically around the processed pixel. For the ThresholdValue hyperparameter, the main purpose was to diminish as many pixels as possible that are false positives. The false positive pixels are the ones that are not in the main object but can be detected by the algorithm due to being in the foreground and having a similar colour to the object. Also, it must be noted that the algorithm is sensitive to the ambient light and the reflecting light can cause some negative issues regarding the spectrum of colours. Considering a ground truth image, labelled by a human, the pixels in the mask (produced by the algorithm) can be split into four main categories:

- True positive: white pixels that are in both images
- True negative: black pixels that are in both images
- False positive: pixels that are black in ground truth but white in the mask
- False negative: pixels that are white in ground truth but black in the mask

Because the algorithm makes a series of dilating operations, the pixels in the False positive category would greatly impact the overall result. Figure 9 shows the impact of certain ThresholdValues with regard to the False positive value.

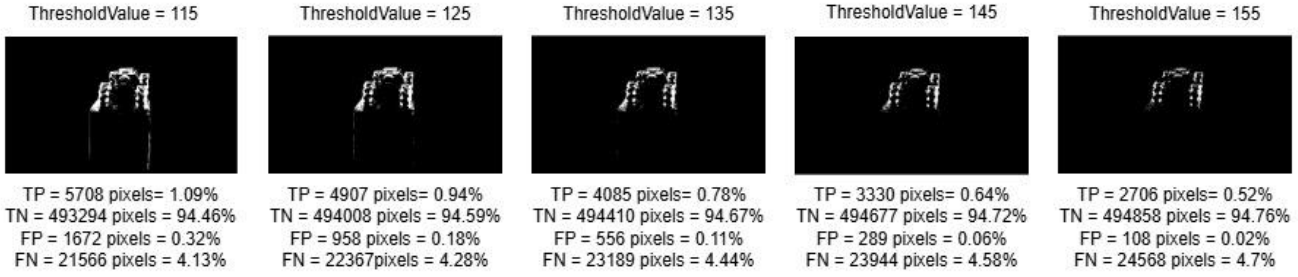

Figure 9. ThresholdValue impact on the percentages of the four categories of pixels

The main evaluation criterion was the mean of the overlap percentages for all the images in the dataset.

As it can be seen from Table 2, the best results were a mean of 89.33% for the overlap area with a corresponding standard deviation of 6.274. These results were obtained for the following values of the hyperparameters: *KernelSize*=5, *ThresholdValue*=135, and *MainPercentArea*=80. Taking into consideration the video and its main object, any changes on the fourth hyperparameter, namely *ThresholdType*, would not make sense because the object's colour had light shades. The time variation is minor and can be attributed to the way in which the scheduling algorithm of the operating system allocates the CPU time for each process running on the computer. The time was measured in seconds using the function perf_counter() from the time module in Python. The algorithm runs using an Anaconda environment in a Windows operating system on an Intel Core I7-8750H CPU. Further analysis of the data in the next subsections will be done under these conditions, given by the best result.

Table 2. Results of Algorithm 1 using different hyperparameter configurations

Experiment's Hyperparameters			**Experiment's Results**				
Kernel Size	Threshold Value	Main Percent Area	Mean Overlap Percent	Standard Deviation	Mean Number Iterations	Worst Overlap Percent	Time (sec)
5	125	80	87.65%	6.469	2.581	67.25%	588.67
5	125	85	87.74%	6.710	2.625	57.80%	589
5	125	90	87.07%	8.633	2.742	30.07%	585.66
5	**135**	**80**	**89.33%**	**6.274**	**3.314**	**67.41%**	**605.75**
5	135	85	89.28%	6.306	3.358	72.83%	638.14
5	135	90	89.02%	7.292	3.440	28.69%	587
5	145	80	88.59%	6.154	4.320	66.46%	594.08
5	145	85	88.71%	5.907	4.364	69.33%	621.22
5	145	90	88.58%	6.036	4.413	69.33%	602.67
5	155	80	87.89%	5.917	5.504	69.09%	583.88
5	155	85	88.18%	5.475	5.563	69.72%	603
5	155	90	88.20%	5.513	5.613	69.72%	597.46
7	125	80	87.00%	7.124	1.874	60.14%	582.81
7	125	85	87.10%	7.151	1.891	60.14%	598.75
7	125	90	86.50%	8.840	1.988	29.72%	589.85
7	135	80	88.66%	6.272	2.363	66.83%	586.04
7	135	85	88.68%	6.327	2.402	72.66%	588.91
7	135	90	88.16%	7.964	2.472	28.37%	591.72
7	145	80	88.23%	6.493	3.076	66.46%	584.91
7	145	85	88.32%	6.375	3.103	69.33%	596.51
7	145	90	88.18%	6.494	3.147	69.33%	592.6
7	155	80	87.61%	5.997	3.833	69.72%	590.39
7	155	85	87.67%	5.773	3.894	69.72%	593.76
7	155	90	87.67%	5.784	3.921	69.72%	582.49
9	125	80	86.11%	7.602	1.569	57.80%	592.73
9	125	85	86.16%	7.623	1.587	57.80%	574.39
9	125	90	85.60%	9.288	1.633	30.07%	636.36
9	135	80	87.69%	6.907	1.924	66.83%	583.72
9	135	85	87.67%	6.913	1.944	71.43%	606.67
9	135	90	87.44%	7.748	1.991	28.05%	598.99
9	145	80	87.46%	6.359	2.408	66.46%	581.6
9	145	85	87.56%	6.183	2.434	67.24%	598.34
9	145	90	87.35%	6.356	2.460	67.24%	591.67
9	155	80	86.92%	5.950	2.985	67.62%	588.55
9	155	85	87.04%	5.748	3.021	67.62%	588.1
9	155	90	87.00%	5.763	3.062	67.62%	587.35

4.3. Local Image Evaluation for Best Setup

In this subsection, we have chosen to highlight the algorithm's results against the ground truth given by the manual labels. The local part of the title is given by the fact that individual images corresponding to the three main cases will be presented:

- Worst labelling, the image with the smallest area of overlap
- Mean labelling, the image with the closest value of overlap to the mean value
- Best labelling, the image with the biggest area of overlap

Figure 10. The images with the bounding boxes for the three main cases. Left: worst case scenario. Middle: the average overlap of the algorithm. Right: best result of the algorithm

Figure 11. The final mask for each of the 3 main cases from Figure 10. Left: worst case scenario. Middle: the average overlap of the algorithm. Right: best result of the algorithm

In the images from Figure 10, the blue rectangle is given by the algorithm, and the red rectangle is the manual labelling. The worst-case scenario (Figure 10 left, Figure 11 left) has an overlap percentage of just 67.41%, with the corresponding image from the database being 305, for an x value of downsampling of 3. The image with a mean labelling efficiency (Figure 10 middle, Figure 11 middle) has an overlap percentage of 89.32%, with the corresponding image from the database being 226. The image with the best result for labelling (Figure 10 right, Figure 11 right) has an overlap percentage of 99.40% and corresponds to the number 40.
As for the masks, the percentage of the biggest continuous white area is as follows:

- For the worst case: 81.57% with 3 dilate transformations needed
- For the mean case: 96.58% with 1 dilate transformation needed
- For the best case: 97.85% with 5 dilate transformations needed

4.5. Global Data Evaluation for the Best Setup

In this subsection, a few global metrics for the entire dataset in the form of histograms and other plots are presented. These graphics will highlight the distribution of overlap percentages, comparison with the normal distribution, and the number of dilate transformations needed for all images in the database.

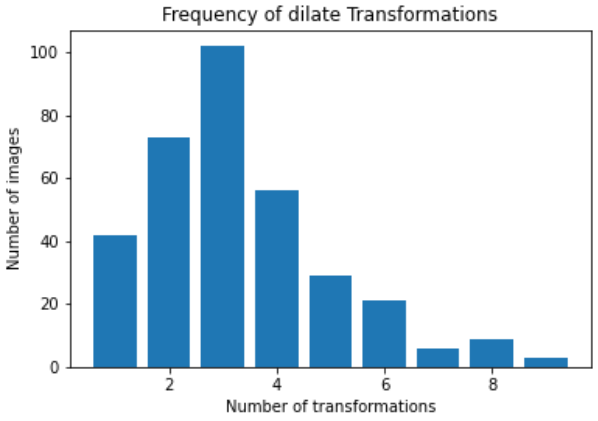

Figure 12. Histogram for the number of dilate operations

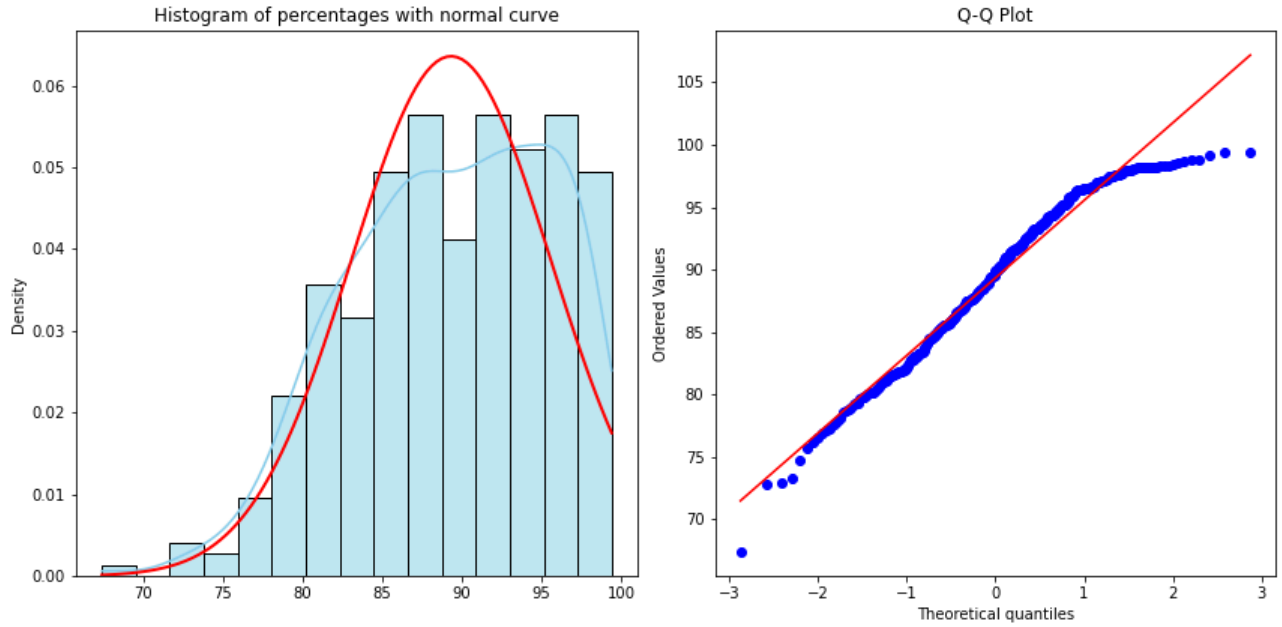

Figure 13. Left: Histogram for the overlap percentages. Right: Quartile to Quartile plot to check the normal distribution of percentages

In Figure 12 is a histogram with the number of dilate transformations needed to reach the minimal percent of the white area for the x-axis and the number of images for the y-axis. As can be seen from the graphic, the majority of the masks, about 100, needed a number of 3 dilate transformations. The number of transformations is inversely proportional to the size of the kernel. For a bigger kernel with a dimension of 7 or 9, the number of dilate operations would be smaller.

In Figure 13 Left, there is also a histogram, but with the values of the overlap percentages of the two rectangles given by the algorithm and the manual label. As can be seen, the worst value is a little under 70% and most values are above 80%. The red line from the graphic represents a normal distribution and the light blue line represents the actual distribution of the data. Using a visual evaluation given by this histogram, we can conclude that the data is not normally distributed, with many values on the higher end, around the value of 100%.

To further analyse if the data of the percentages is normally distributed, we have employed a quartile-quartile plot in Figure 13 Right. This plot creates two sets of quartiles, one for the normal distribution, the red line, and the other for the actual data, the blue dots. If the blue dots come from the same distribution as the red line, this would mean that they should align more with the red line (Ford, 2015). This is not true for the higher values, the ones close to 100%. So, this graphic also informs us that the percentages are not from a normal distribution.

5. Limitations and Challenges

One of the main challenges is the processing time needed by the algorithm to eliminate the background and to distinguish between foreground objects. As was established in Table 2, the values of the hyperparameters do not change too much in the overall time taken by the algorithm. The small difference in time is a result of the operating system's scheduling algorithm. For a small dataset, this issue (time taken by the algorithm) is not relevant, but with the increase in the number of images being processed, time can become a problem.

To analyse the scalability of the proposed architecture in the context of the dataset used in this article, I have repeated the same experiment, with the same values of the hyperparameters but with a different dataset size. The results of the experiment are presented in Table 3.

Table 3. Impact on the processing time for different sizes of the dataset

Number of images	50	100	150	200	250	300	341
Time in seconds	84.22	165.82	247.75	340.49	426.1	513.62	585.16

As shown in Table 3, the time increases directly proportional to the number of images. This behaviour is normal, but it can become a problem for datasets with tens of thousands of images. One option to mitigate the impact of time on the algorithm is to reduce the number of image writes by skipping the optional debugging and tracking mechanisms. Another option suitable for this dataset is to calculate if some images could be obtained from previous ones based on geometrical transformation, so that the same mask could be applied multiple times with minor changes.

Another limitation of the algorithm is the ability to detect only one object, the main object from the foreground. To better emphasise this problem, Figure 14 presents two images, on the left is only one car, and on the right, there are 2 cars, from which only one of them is detected.

Figure 14 Left: Single car detection. Right: Detecting the main car from 2 vehicles

6. Conclusion and Future Development

In this paper, we put forward a model of constructing a database of images and using a combination of CNN architecture and morphological operations in order to identify the region of interest of the main object of the video. Furthermore, the algorithm saves the coordinates of the detected bounding box that best frames the object using a YOLO format, thus creating a folder of labels for each image. To verify how well the algorithm performs, the paper compares manual labelling with the results given by the program. For the process of comparison, we established a metric given by the percentage of overlap between the bounding box given by the manual labelling and the bounding box given by the algorithm.

In terms of future developments, we intend to implement an adaptive kernel that will be able to modify its dimensions based on the iteration number, the percentage of the biggest continuous white area from the total white area, and the distance between the two largest white areas. We would also like to extend the algorithm's capabilities to be able to identify more objects and classify them into a certain number of classes using unsupervised learning. This technique implies that each object would be assigned a number and for each region of interest detected in the image, a specific number corresponding to the most similar detected object would be associated with that label. Also, on this topic, the number of objects could be predefined, or it could increase based on a predefined threshold similarity score and a comparison of the current object with the objects that were already discovered.

References

Amazon. (n.d.). What is data labeling? *Amazon Web Services (AWS).* Retrieved from https://aws.amazon.com/what-is/data-labeling/

Bhargavi, K., & Jyothi, S. (2014). A survey on threshold-based segmentation technique in image processing. *International Journal of Innovative Research & Development, 3*(12), 234–239. Retrieved from https://www.researchgate.net/publication/309209325_A_Survey_on_Threshold_Based_Segmentation_Technique_in_Image_Processing

Caporali, A., Pantano, M., Janisch, L., Regulin, D., Palli, G., & Lee, D. (2023). A weakly supervised semi-automatic image labeling approach for deformable linear objects. *IEEE Robotics and Automation Letters, 8(2)*, 1013-1020. https://doi.org/10.1109/LRA.2023.3234799

Das, M. (2023). Rembg: Effortlessly remove backgrounds in Python. *Medium.* Retrieved from https://medium.com/@HeCanThink/rembg-effortlessly-remove-backgrounds-in-python-c2248501f992

Educative. (n.d.). What is image thresholding? Retrieved from https://www.educative.io/answers/what-is-image-thresholding

Fenchel, M., Thesen, S., & Schilling, A. (2008). Automatic labeling of anatomical structures in MRFastView images using a statistical atlas. *MICCAI Medical Image Computing and Computer-Assisted Intervention*, 576-584. https://doi.org/10.1007/978-3-540-85988-8_69

Ford, C. (2015). Understanding QQ plots. *University of Virginia Library*. Retrieved from https://library.virginia.edu/data/articles/understanding-q-q-plots

Gatis, D. (n.d.). *Rembg*. GitHub. Retrieved from https://github.com/danielgatis/rembg

Haselhoff, A., Schauland, S., & Kummert, A. (2008). A signal theoretic approach to measure the influence of image resolution for appearance-based vehicle detection. *Proceedings of the 2008 IEEE Intelligent Vehicles Symposium*, 822–827. https://doi.org/10.1109/IVS.2008.4621252

He, X., Zemel, R. S., & Carreira-Perpinan, M. A. (2004). Multiscale conditional random fields for image labeling. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/CVPR.2004.1315232

Hugging Face. (n.d.). Dataset card for ImageNet. Retrieved from https://huggingface.co/datasets/ILSVRC/imagenet-1k

Ji, H., He, J., & Yang, X. (2014). Scale-invariant auto-context for object segmentation and labeling. *KSII Transactions on Internet and Information Systems, 8*(8), 2881-2894. https://doi.org/10.3837/tiis.2014.08.018

Jia, X., Tong, Y., Qiao, H., Li, M., Tong, J., & Liang, B. (2023). Fast and accurate object detector for autonomous driving based on improved YOLOv5. *Scientific Reports, 13*, 9711. https://doi.org/10.1038/s41598-023-36868-w

Khan, S. A., Lee, H. J., & Lim, H. (2023). Enhancing object detection in self-driving cars using a hybrid approach. *Electronics, 12*(13), 2768. https://doi.org/10.3390/electronics12132768

Khanam, R., & Hussain, M. (2024). YOLOv11: An overview of the key architectural enhancements. *arXiv*. https://doi.org/10.48550/arXiv.2410.17725

Makridakis, S. (2017). The forthcoming artificial intelligence (AI) revolution: Its impact on society and firms. *Futures, 90*, 46-60. https://doi.org/10.1016/j.futures.2017.03.006

Marr, B. (2023). 15 amazing real-world applications of AI everyone should know about. *Forbes*. https://www.forbes.com/sites/bernardmarr/2023/05/10/15-amazing-real-world-applications-of-ai-everyone-should-know-about/

Neufeind, M., O'Reilly, J., & Ranft, F (2018). *Work in the digital age: Challenges of the fourth industrial revolution*. Rowman & Littlefield International Ltd.

OpenCV. (n.d.). Morphological transformations. Retrieved from https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

Qin, X., Zhang, Z., Huang, C., Dehghan, M., Zaiane, O. R., & Jagersand, M. (2020). U2-Net: Going deeper with nested U-structure for salient object detection. *Pattern Recognition, 106.* https://doi.org/10.1016/j.patcog.2020.107404

Rahman, R., Azad, Z. B., & Hasan, M. B. (2022). Densely-populated traffic detection using YOLOv5 and non-maximum suppression ensembling. *Proceedings of the International Conference on Big Data, IoT, and Machine Learning, 95,* 567-578. https://doi.org/10.1007/978-981-16-6636-0_43

Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (pp. 843–852). IEEE. https://doi.org/10.1109/ICCV.2017.97

Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). YOLOv10: Real-time end-to-end object detection. *arXiv*. https://doi.org/10.48550/arXiv.2405.14458

Wang, C.-Y., Yeh, I.-H., & Liao, H.-Y. M. (2024). YOLOv9: Learning what you want to learn using programmable gradient information. *arXiv*. https://doi.org/10.48550/arXiv.2402.13616

Wu, W., & Yang, J. (2009). Semi-automatically labeling objects in images. *IEEE Transactions on Image Processing, 18(6)*, 1340-1349. https://doi.org/10.1109/TIP.2009.2017360

Zhang, L., Tong, Y., & Ji, Q. (2008). Active image labeling and its application to facial action labeling. *European Conference on Computer Vision (ECCV), 5303*, 706-719. https://doi.org/10.1007/978-3-540-88688-4_52

Zhang, W., Hu, H., Hu, H., & Yu, J. (2020). Automatic image annotation via category labels. *Multimedia Tools and Applications, 79*, 11421-11435. https://doi.org/10.1007/s11042-019-07929-y