Public transport vehicle detection based on deep learning

Comănac Dragoș-Mihail
Faculty of Mathematics and Computer Science
Babeș-Bolyai University
Cluj-Napoca, Romania
comanacdragosmihail@gmail.com

***Abstract*—Billions of people that suffer from some form of visual impairment, out of which a significant part is legally blind. Also, a basic human need is mobility, but there aren't enough traditional mobility solutions for all visually impaired persons, such as assistance dogs, thus, for most legally blind people this need can't be easily satisfied. A more scalable solution would be a digital one, which involves computer vision.**

Therefore, the main purpose of this paper is to provide a form of mobile assistive technology, based on object detection for visually impaired persons.

Our object detector is implemented along the lines of You Only Look Once. We train on a subset of Open Images V4 dataset composed of bus, car, and license plate, a single convolutional neural network. Also, we have developed an Android mobile application that uses this object detector in order to visualize the bounding box predictions. The key feature of the application is the accessible live object detection, in which the predictions are converted to sound and played using the mobile device speakers.

We obtain a mean average precision of 70.03%, and for the bus class, we obtain an average precision of 90.01%, for the car class 64.04% and for the license plate 56.68%, with a speed of around 5 FPS on a mobile device.

***Index Terms*—object detection, deep learning, convolutions, YOLO, visual impairment**

I. Introduction

According to the World Health Organization [14], the number of people suffering from some moderate to severe form of distance vision impairment or blindness due to cataract or uncorrected refractive error is around 200 million, out of the total of 2.2 billion people worldwide that are estimated to have problems with their vision. This represents a significant segment of the population that has trouble performing daily tasks. These troubles can be alleviated using assistive technologies that can help persons with disabilities maintain or enhance their capabilities. Given the number of people that suffer from some form of visual impairment and the fact that computers can substitute visual functionalities, computer vision has the potential to play the main part of assistive technology for visually impaired persons (VIP), such that it helps the user to better understand the surrounding environment when performing different kinds of tasks.

One of the fundamental human needs is mobility and it can be achieved through public transport. This way of traveling is especially important to the VIP, since they can't drive. Therefore, the main ways a VIP can travel is by public transport, ridesharing, or taxi, but they experience many difficulties on their journeys, often experiencing social exclusion because they are limited in their choices of public transport [12].

Given that most mobility solutions don't provide adequate accessibility facilities, we developed **a mobile assistive technology solution** for VIP that provides spatial information by using a **real-time object detection model** inspired by You Only Look Once [15]. More specifically, the user can get information about buses or cars from the auditory information provided by the mobile application. All this information is extracted from the bounding boxes predicted by the object detection model. Also, information about the license plates can be provided to help the user to identify the vehicle.

The main contributions of this work are two-fold: first we proposed a pipeline to improve VIP access to public transport. We start by taking live images and output the bounding boxes in the form of audio, on the mobile device. Secondly, we implemented from scratch and trained using fine-tuning an object detector inspired by YOLOv2 [16]. For this process we performed in-depth data analysis and we enhanced the train datasets with the locations of license plates using an off-the-shelf, state of the art object detector [7]. Finally, we experimented with several augmentation techniques. To this end, we obtain a mean average precision of 70.03%, and for the bus class, we obtain an average precision of 90.01%, for the car class 64.04% and for the vehicle registration plate 56.68%, with a speed of around 5 FPS on a mobile device.

In what follows, we first describe the current approaches in terms of public transport accessibility and object detection in Section II. Afterwards, in Section IV we present the implementation details of our approach by detailing our object detector. Then, we discuss the performance of our object detector and how it is compared to other object detection systems in Section V. Finally, we briefly present how we deploy our object detector on a mobile phone in Section III.

II. State of the art

We emphasize what are the current options the VIPs have when it comes to public transportation. Also, we will review some of the most performing object detection systems that are currently used in various domains.

A. Accessible public transportation applications for VIP

Broadly speaking, there are two options of improving public transportation for VIPs: on one hand, classical methods which are based on radio signals, and on the other hand, the more lightweight solutions based on computer vision.

1) Radio frequency approach: The current approach to making buses more accessible to the VIP is a solution based on Radio Frequency Identification (RFID) [4]. Basically, it uses wireless radio frequency transmissions to transfer data between two devices. The disadvantage of this method is the complex infrastructure required that makes the bus, station, and the user dependent upon each other.

2) Computer vision approach: Another approach would be to make the user independent of any infrastructure. This can be achieved by using a mobile application that uses an object detection model that tells the user where the bus is located by using real-time object detection.

Using this approach, Travis [18] is an Android add-on that is simply connected to the smartphone and uses its computing power to execute several computer vision tasks, mainly object detection in order to provide information about public transport and the surrounding environment.

B. Object detectors

Initially, the task of object detection was decomposed into multiple tasks, that together made a pipeline, which is hard to train. For example, region-based object detectors such as R-CNN [6] and its faster variants first generate bounding boxes through selective search, then a convolutional network extracts features that are classified by a support vector machine. All these steps slow down performance. This approach is called two-stage object detection.

Single-shot object detectors achieve real-time speed with decent accuracy [10], [15] because their detection pipeline consists only of one neural network that processes the image and directly output the predictions. This approach used to have low accuracy, but recent advances have made the single-shot detectors rival the two-stage detectors in terms of accuracy, without losing speed. Thus, we will focus especially on the single-shot detector class of object detection models.

You only look once (YOLO) [15] is a single-shot object detection system that achieves real-time performance. Instead of a long and complex detection pipeline, in YOLO, the object detection problem is treated as a regression problem. There is only one neural network that predicts the bounding boxes and class probabilities from an image. This way, the network can benefit from using end-to-end learning, and the inference time is greatly reduced, thus achieving real-time performance.

The network is composed of two parts. The first one consists of what is called the base network, which is a truncated version of an image classifier, where the classification layers are removed, that is used to extract features. On top of the base network, a head is added that output the prediction in the form of a grid with multiple anchors for each cell that predict a bounding box each.

In order to see how accurate the model is, mean average precision (mAP) is used. Firstly, the predictions are sorted descending by their confidence score. Then, the predictions are parsed one by one, and at each step, the precision and recall are computed, taking into consideration only the parsed prediction up until that point. Average precision is defined as the area under the precision-recall curve and mAP is defined as the mean of the average precisions for each class.

According to [15], in terms of performance on the Pascal VOC 2007 dataset [5], the first version of YOLO achieves 63.4% mAP and 45 frames per second (FPS), and the faster version has 52.7% mAP and 155 FPS. For comparison, Faster R-CNN [17] achieves 73.2% mAP but 7 FPS, which is accurate but very slow, and the Deformable parts model [20] achieves 100 FPS but it has 16% mAP, or a slower variant achieves 26.1% mAP at 30 FPS. Therefore YOLO strikes a good balance between speed and accuracy.

Newer versions of YOLO bring some incremental improvements. Also, the Microsoft COCO dataset [24] is another relevant dataset for comparing object detection systems. On this dataset, YOLOv4 [1] achieves 43.9% mAP at 31 FPS, or a smaller variant achieves 38 FPS, with 41.2% mAP.

Single Shot MultiBox Detector (SSD) [10] is an object detector that achieves real-time performance, encapsulating all operations in a single deep neural network

Similar to YOLO, the network is composed of two parts. The key features of SSD are the multi-scale feature maps, convolutional predictors, and default boxes.

Several convolutional layers are appended to the base network that decrease progressively in size the feature map from the base network. This way predictions are made for each newly added layer, therefore, the predictions are made at various scales.

Regarding performance on the Pascal VOC 2007 dataset, SSD achieves 74.3% mAP and 59 FPS, surpassing the first version of YOLO in terms of speed and accuracy balance.

On the Microsoft COCO and in the context in which YOLOv4 was tested, described in [1], SSD achieves 43 FPS with 25.1% mAP. This shows that YOLO surpasses SSD in terms of prediction quality, with minimal time costs.

III. Proposed solution

In Fig. 1 we present the main modules of our application. This is a proof of concept for a mobile assistive technology that uses a model of object detection with deep learning, deployed on an Android application. The application can perform object detection on static images and on the live feed from the mobile device's camera, but our main contribution is the accessible live object detection in which the bounding boxes are not drawn on an image, but are converted to text and played on the mobile device speakers. In this way, a VIP could use the application in order to gather information about the environment. We also use an OCR API to provide extra information about the text found in the image, such as the license plate.

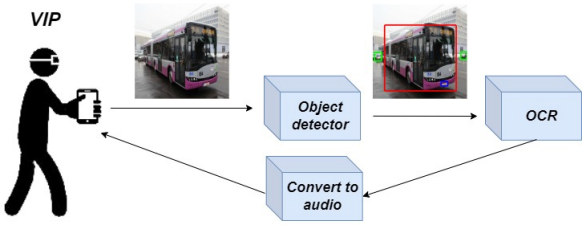


Fig. 1: Step by step pipeline of our solution

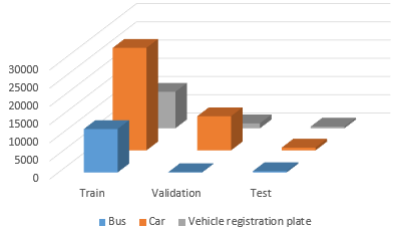


Fig. 2: Undersampled and enhanced dataset bounding boxes distribution

The object detection system is run using the TensorflowLite Task Library, more specifically the object detector that it's already implemented. In order to use this class, the model must be stored in a .tflite file and it must meet some compatibility requirements regarding the input and the output of the model, which are described on the official website [22].

IV. Object detector

In this section we present the details of our object detection system implementation[1], which is in Python 3.7 and uses Tensorflow 2.3, including Keras.

A. Dataset

For training the object detection system we use the Open Images Dataset V4 [9], available at [8]. In total, the dataset contains 9.2 million images, including 14.6 million bounding boxes across 600 classes on 1.74 million images. We use only a subset of classes: bus, car, and license plate or vehicle registration plate as it is called in the original dataset.

The tool used to download the bus, car, and license plate classes and the corresponding bounding boxes is OIDv4 ToolKit [23]. The dataset is split into three parts: train (77.81%), validation (19.54%), and test (2.65%).

The dataset, as it is in its original form, is unbalanced. The car class has around 5-6 times the number of bounding boxes the other classes have. This is problematic because the object detector tends to predict mostly cars.

We try to address this issue by using a technique called undersampling. The idea is that we try to balance the numbers by removing instances of the dominant class.

We further improve the dataset by enhancing the license plate class using an existing highly performant license plate detector. Therefore, we use an object detector based on YOLO [7] in order to add new bounding boxes if they don't already exist, because we observed that license plates are not annotated in a lot of images. We can see in Fig. 2, the number of boxes and images for each class, and in Table. I we can see the detailed distribution. In total, we have added 3108 license plate bounding boxes to the dataset.

Each image is resized so that its dimension is 416×416 and the bounding boxes are scaled accordingly. This is done using linear interpolation.

The resizing helps the object detection task, as explained in [16] because this way we can split the image into a grid of 13×13 cells of size 32×32 pixels so that there is a cell in the center that can detect the larger objects that are centered in the middle, rather than have 4 cells in the middle that try to detect the same object.

Each image is associated with multiple bounding boxes and each cell is responsible for detecting multiple bounding boxes through the use of anchors as explained in [16]. We use three anchors per cell so that each cell can detect various shapes and sizes. The anchor boxes are chosen using K-Means over the dataset.

For computing the distance between the centroid and bounding box the following formula is used, as in [16]:

$$distance(centroid, box) = 1 - IOU(centroid, box)$$

Where IOU represents intersection over union between the centroid and the bounding box.

Each image needs to be associated with a ground truth. We represent the ground truth as a $C \times C \times B \times (5 + N)$ tensor. For each cell in the $C \times C$ grid, we associate B anchor boxes. Each anchor box has the following parameters: b_x and b_y represent the center of the box, b_w and b_h represent the width and height, c is the probability that the anchor predicts an object and $c_1, c_2, c_3, ...c_n$ represent the class probabilities. In our case, C is 13 and both B and N are 3. Therefore, the output is a tensor of shape $13 \times 13 \times 3 \times 8$.

The formulas that are used to compute the center, width, and height from the output of the model are:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w \cdot e^{t_w} \\ b_h &= p_h \cdot e^{t_h} \end{aligned} \tag{1}$$

Where t_x, t_y, t_w, t_h represent the raw predictions to which the sigmoid function is applied, c_x, c_y represent the coordi-

TABLE I: Bounding box statistics related to the undersampled and enhanced subset of Open Images

	No. images	No. bounding boxes			
		Total	Bus	Car	License plate
Train	19773	50110	11927	28159	10024
Validation	4967	10824	92	9381	1351
Test	669	1580	353	764	463

[1] *Full implementation on GitHub*

nates of the upper left corner of the cell that predicts the box, and p_w, p_h represent the width and height of the anchor that predicts the box. Over the output for the class probabilities, the softmax function is applied.

B. Model

The object detection model is inspired by YOLOv2 [16]. The neural network is fully convolutional and is composed of three parts: feature extractor, neck, and head. For the feature extractor, we use MobileNetv2 [21] because of its flexibility and the fact that it uses depthwise convolutions, inverted residual blocks, and linear bottlenecks, which all help with performance, thus consuming less power, which is crucial for mobile solutions. Also, we use pretrained weights on ImageNet [2] to benefit from transfer learning.

For the rest of the model, we use convolution blocks which are composed of a convolution layer that uses HeNormal initialization, batch normalization, and optionally LeakyReLU activation function. In general, we choose an alpha of 0.1 for LeakyReLU.

The neck is inspired by U-Net [19]. The aim is to add features from earlier layers to the result through skip layers and upsample blocks that use transposed convolutions, followed by LeakyReLu and batch normalization. This helps the network to "see" the image at multiple resolutions as explained in the fine-grained features section in [16].

On top of the upsamlple blocks, a dropout layer is used for regularization in order to reduce overfitting. Another reason for adding this layer at this specific position is that the upsample blocks have the most trainable parameters, compared to other areas of the model. After the dropout layer, a convolutional block and two inverted residual blocks are added, which help in refining the feature maps.

The head is composed of a convolution layer that has 24 filters in our case, so the final output is $13 \times 13 \times 3 \times 8$ after a reshape layer. This is because we use three anchors and three classes. This can vary if other datasets are used.

Also, an additional input that represents all the true bounding boxes is directly added to the output. This is an implementation trick that only helps in the computation of the loss because, even though each image is associated with a specific anchor, it is not restricted to be predicted only by that anchor. If the IOU threshold between the predicted box from another anchor and one of the true bounding boxes is high enough, that prediction is considered correct. During normal inference, a dummy array is passed for this input.

In total, our model has around 2 million parameters, out of which around 1.3 million are from MobileNet.

C. Loss

During training, we optimize a composed loss function adapted from [13]:

$$L = L_{loc} + L_{obj} + L_{class}$$

The first component is basically a sum-squared error, handling the localization loss:

$$L_{loc} = \frac{\lambda_{coord}}{N_{L^{obj}}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} \cdot [(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 +$$

$$+ (\sqrt{w_{ij}} - \sqrt{\hat{w}_{ij}})^2 + (\sqrt{h_{ij}} - \sqrt{\hat{h}_{ij}})^2]$$

Where $L_{ij}^{obj} = \begin{cases} 1 & C_{ij} = 1 \\ 0 & otherwise \end{cases}$ is an indicator function in which C_{ij} means that there is an actual object in the i'th cell and j'th anchor. $N_{L^{obj}}$ just represents the number of actual object in the image and it is given by the following formula: $\sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj}$. The center of the bounding box is denoted using the x for the horizontal position and y for the vertical position. The width is denoted with w and the height with h. The square root of the width and height is used because, otherwise, the error in small and large bounding boxes is treated the same.

The second component is related to the objectness of a bounding box, which represents the probability that an object is present in that bounding box:

$$L_{obj} = \frac{\lambda_{obj}}{N^{conf}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} (IOU_{prediction_{i,j}}^{ground\ truth_{i,j}} - \hat{C}_{ij})^2 +$$

$$+ \frac{\lambda_{noobj}}{N^{conf}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{noobj} (0 - \hat{C}_{ij})^2$$

Where $L_{ij}^{noobj} = \begin{cases} 1 & max_{i'j'} IOU_{pred_{i,j}}^{GT_{i',j'}} < IOU_t \text{ and } C_{ij} = 0 \\ 0 & otherwise \end{cases}$ is an indicator function which is one only if a predicted bounding box that does not appear in the ground truth in the respective cell and anchor, has the IOU overlap with any ground truth bounding box less than IOU_t. Basically, if the prediction has an IOU overlap with any bounding box larger than IOU_t, but it does not appear in the ground truth, then it is considered correct and it's not penalized, otherwise, it is not considered an object and it must increase the loss. Here we use the extra output explained in the previous section. $N^{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} + L_{ij}^{noobj}(1 - L_{ij}^{obj})$ counts the number of bounding boxes from the ground truth, but also the boxes that predict objects where there shouldn't be any. Therefore, the first part penalizes the errors in the confidence scores for objects that should be predicted, and the second part penalizes the boxes that predict an object that should not be there.

The last component represents the loss from the class probabilities, and when multiple classes are involved, usually cross-entropy loss is used:

$$L_{class} = -\frac{\lambda_{class}}{N_{L^{obj}}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} L_{ij}^{obj} \sum_{c \in classes} p_{ij}^{c} log(\hat{p}_{ij}^{c})$$

Most grid cells do not contain any boxes. Therefore, in order to balance the confidence scores, $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$

are added in order to increase the loss from bounding box predictions and decrease the loss from confidence predictions. Also, $\lambda_{obj} = 2$ and $\lambda_{class} = 3$ are added to control the loss from the objectness and class losses. For IOU_t we choose a value of 60%.

D. Data augmentation

We apply various photometric data augmentation techniques such as random hue (Fig. 3b), random brightness (Fig. 3c), random contrast (Fig. 3d) and random saturation (Fig. 3e). The data augmentation techniques in Fig. 3 are applied over Fig. 3a.

In Fig. 4 we present other data augmentation techniques that we have used. In Fig. 4a we represent the Cutout technique, introduced in [3]. The idea is to make the pixels from a random patch in the image black. This way the network should adapt to recognize objects even if they are partially visible. We also follow the recommendation from [3], which states that the patch doesn't have to fit fully in the image. This means that if the center of the patch falls near one of the edges of the image, only the part that overlaps with the images is blacked out.

The other technique is called Mosaic, Fig. 4b, and it was first introduced in [1]. Here, four images are concatenated in order to form a single image. The effect is that the number of bounding boxes in a single image is increased, and the size of the bounding boxes becomes smaller, thus the network should better adapt to small bounding boxes.

In practice, for each image, we apply a random data augmentation technique from Fig. 3, and we apply all these data augmentation techniques only during training, and all of them, except Mosaic, are vectorized in order to be computed on the GPU.

E. Training

For training, we use a pretrained version of MobileNetV2 [21] on ImageNet [2], and for the optimizer we use Adam. We use a cosine annealing scheduler for the learning rate, described in [11], which follows the following formula:

$$LR_{epoch} = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \cdot (1 + cos(\frac{epoch}{T} \cdot \pi)) \quad (2)$$

The main idea is that the learning rate will follow a cosine wave between η_{min} and η_{max}. The epoch represents the current epoch, and T represents the restart epoch, meaning that every T epochs the monotony is changed, and the learning rate will reach either η_{min} or η_{max}.

(a) Original (b) Hue (c) Brightness (d) Contrast (e) Saturation

Fig. 3: Photometric data augmentation techniques

(a) Cutout (b) Mosaic

Fig. 4: Other data augmentation techniques

We train the model for 50 epochs on a GPU using early stop with the patience of 5 epochs and a delta of $1e^{-4}$. This means that if the model does not improve after some epochs, by the given delta, the training stops because we don't want to overtrain, in order to both save time and reduce overfitting also.

At the beginning of the epoch, the learning rate is set, following the cosine annealing formula, and at the end, the images are shuffled in order for the network to see the images in a different order, each epoch. Also, during training, before any processing, random photometric data augmentation is used, followed by cutout and mosaic as explained in a previous section.

During a normal training session, the weights of the pretrained model are frozen, meaning that they do not update. We do this in order to not break the knowledge stored in the weights. But, during a fine tuning session, which occurs after a normal training session, we set a very small learning rate and unfreeze the pretrained model. By doing this, the previously frozen weights are updated to better fit our dataset. Usually, a fine tuning epoch takes much longer, because the pretrained model has the largest share in parameters of the total number of parameters.

In Fig. 5 we present the evolution of the loss during training for our best model. Ideally, the validation loss is always around the training loss. This is a sign that the model does not overfit.

F. Inference

The inference represents a pipeline of processing an input and getting the predictions for that input. In our case, the input is an image and there are three stages of processing that an image goes through in order to get the output, represented by the bounding boxes.

Firstly, there is preprocessing, which consists of normalizing the images in the range [-1, 1]. This is required by the Mo-

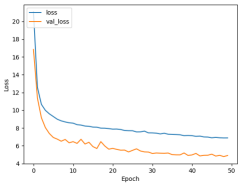


Fig. 5: Training (blue) and validation (orange) loss for our best model

bileNetV2 backbone model. Also, here the data augmentation occurs, before the normalization.

The second step is passing the image through the actual neural network.

Postprocessing is the final step. Firstly, the bounding boxes are extracted from the resulting tensor of size $C \times C \times B \times (5+C)$. Basically, for each cell in the $C \times C$ grid, we extract the $B \cdot (5+C)$ raw bounding boxes. Then the raw values are converted using 1 to obtain the actual values. These boxes are filtered based on their score.

Usually, there are a lot of overlapping boxes that predict the same object. This is solved using Non-maximum Suppression (NMS) which prunes away extra bounding boxes by ordering the boxes by their scores, descending. Then each box is kept only if they have a low enough IOU with any previously kept bounding box with the same label. In this way, if there are a lot of boxes with the same label in some area, only the one with the highest score is kept.

V. Experimental results

A. Results

In this section we present our final results on the subset of Open Images Dataset V4 [9], comprising three classes: bus, car, and vehicle registration plate.

To compute the exact improvements, we consider the maximum value from the mAP curve in Fig. 7. We detail the exact improvements after hyperparameter tuning, in percents, in Table II.

TABLE II: Improvement after hyperparameter tuning in percents for each class in AP and mAP for the average case

Class	Car	Bus	License plate	Average
Before tuning	59.52%	84.07%	51.25%	64.95%
After tuning	62.54%	90.23%	55.09%	69.29%
Improvement	3.02% ↑	6.16% ↑	3.84% ↑	4.34% ↑

After tuning the hyperparameters, we perform fine tuning, meaning that we unfreeze the backbone of the model and further train with a very small learning rate. For fine tuning, we have used $\eta_{max} = 10^{-5}$ and $\eta_{min} = 10^{-8}$.

In Fig. 7 we can see that we achieved small improvements only by fine tuning and in Table III we detail the exact values obtained and the improvements. For the bus class there is a slight decrease, but in general the improvements are positive.

In terms of speed, our solution achieves approximately 5 FPS on a Samsung A70 mobile phone.

We present the result on some images from Cluj-Napoca in Fig. 6. Green bounding boxes represent cars, red bounding boxes represent buses, and blue bounding boxes represent registration plates.

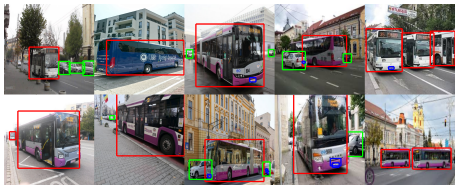


Fig. 6: Images from Cluj-Napoca

TABLE III: Improvement after fine tuning in percents for each class in AP and mAP for the average case

Class	Car	Bus	License plate	Average
Before fine tuning	62.54%	90.23%	55.09%	69.29%
After fine tuning	64.04%	90.01%	56.68%	70.03%
Improvement	1.5% ↑	0.22% ↓	1.59% ↑	0.74% ↑

B. Hyperparameter tuning

Hyperparameters are important because the performance of an algorithm can be improved by simply tuning the hyperparameters. We select some hyperparameters and explore how they impact the mAP. Also, in order to better comprehend the differences, we compute the mAP at different score thresholds, which results in a curve that we use to compare different models. In general, we use a true positive threshold of 50% and a NMS threshold of 30%.

We start by analyzing the scheduler. In the formula from 2, we use for the maximum learning rate a value of 10^{-3} and for the minimum learning rate a value of 10^{-6}. We compare how different values for the restart epoch parameter T influence the mAP. We detail in Table IV the value for T used in training each model. In general, the mAP value in the table is the maximum on the mAP curve across various scores.

We can tell from Table IV that a value of 60 yields the best results. These models were trained for 50 epochs, and we can see that in general, values under 50 for the restart epoch give slightly worse results, meaning that increasing and decreasing the learning rate doesn't help that much. Therefore, further on, we use 60 as the restart epoch.

TABLE IV: Restart epoch values

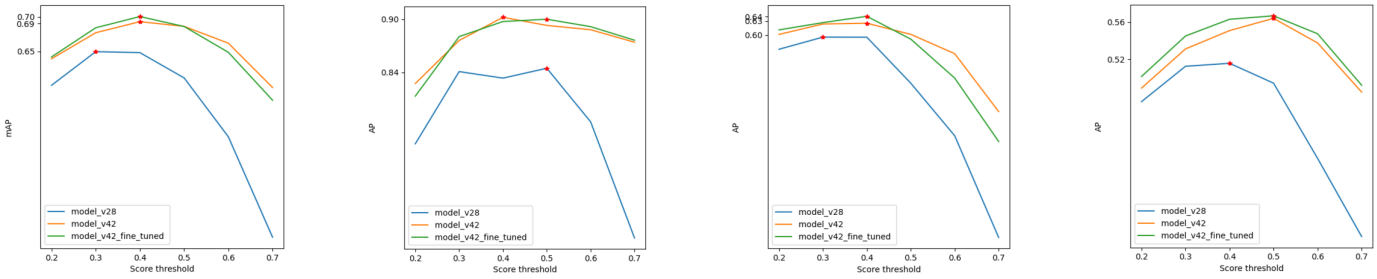
Model	v29	v30	v31	v32	**v33**	v34
T	100	50	25	10	**60**	75
mAP	67.7%	67.17%	64.23%	66.36%	**67.72%**	65.74%

Next, we see how the batch size influences the performance. In V we detail the batch size value used in training each model. In our case, the smallest batch size yields best results. Usually, a larger batch size should give better results, but our dataset is relatively small, and this could be a reason why a small batch size is better. Our model is also very small, thus it would benefit from more precise changes given by a smaller batch size.

TABLE V: Batch size values

Model	v33	v35	**v36**
Batch Size	32	16	**8**
mAP	67.72%	62.88%	**68.91%**

The dropout probability is important because of its regularization effect. We study various values presented in Table VI. We go further on with a value of 30% because it is the best in the average case.



(a) mAP (b) Bus (c) Car (d) License plate

Fig. 7: AP and mAP for the final model (orange), the model before hyperparameter tuning (blue), and the model after fine-tuning (green)

TABLE VI: Dropout values

Model	**v36**	v37	v38	v39
Dropout	**30%**	40%	50%	20%
mAP	**68.81%**	65.64%	68.22%	67.33%

So far, we have tuned parameters related to training or to the model itself. Finally, we will see how the data augmentation hyperparameters affect the performance.

For the cutout data augmentation, we consider as a hyperparameter the length of the side of the cutout square. We can see in Table VII the different values that we have chosen to see the influence of the size of the cutout patch. The lowest and the highest values yield the worst results, but increasing the cutout until 192×192 also increases the mAP. Further on we use this value.

TABLE VII: Cutout values

Model	v39	v40	v41	**v42**	v43
Cutout	64	32	128	**192**	256
mAP	67.33%	63.03%	68.79%	**69.29%**	59.33%

For the mosaic data augmentation, we study the influence of the probability that it is applied and the minimum size that one of the four images can take. For example, when the minimum size is 50, then each image will have a size of at least 50×50. In Table VIII we describe the values for the minimum size, where a value of 50 gives the best results, and the lowest and the highest values give the worst results.

TABLE VIII: Mosaic minimum size values

Model	**v42**	v44	v45
Size	**50**	100	25
mAP	**69.29%**	66.25%	67.54%

In Table IX we have the values that we have used for the probability that mosaic data augmentation is applied during training. We can see that a low mosaic probability such as 40% results in low mAP. In the average case, the models with 60% and 80% probability are close in terms of mAP, but because the mAP for the bus class is much larger with a probability of 80% we consider this value to be the best.

TABLE IX: Mosaic probability values

Model	**v42**	v46	v47
Probability	80%	60%	40%
mAP	69.29%	**69.6%**	68.11%
Bus AP	**90.23%**	88.16%	87.98%

In Table X we present the hyperparameters used in the data augmentation techniques presented in Fig. 3.

TABLE X: Photometric data augmentation hyperparameters

Random Hue	delta	0.5
Random Saturation	lower	5
	upper	10
Random Brightness	delta	0.3
Random contrast	lower	1
	upper	2

We have tried to use three, four and five centroids, or clusters, in order to generate the anchors, but in all cases the outer boxes are the same and in the case of four and five anchors, only smaller and smaller anchors are added which are not different enough from each other, therefore we choose to use the variant with three anchors.

C. Comparison with other methods

State of the art object detection models such as YOLOv4 [1] achieve real-time performance and overall good detection accuracy. We use the COCO dataset [24] in order to compare our method with other existent methods. This dataset has several splits such as validation or test-dev. We test our method by performing object detection on each of these dataset splits and uploading the results to the evaluation server. We follow the guidelines regarding the data format described on the official website, where details about the evaluation server are also provided.

On this dataset, the YOLO state of the art detector achieves 43.9% mAP, SSD achieves 25.1% mAP, and the currently best mAP on the detection competition leaderboard is 63%. Our method achieves a modest 0.4% mAP on the test-dev bounding box evaluation server, indicating that there is still room for improvement. This performance might be due to the fact that our method was developed considering only three classes, and the COCO dataset has 80 classes. We also tried to

develop a very small model because the resources are limited on a mobile device. For example, our model has around 14 megabytes, whereas better object detectors have around 200-300 megabytes or more. Also, the hyperparameters are not tuned for this specific dataset.

Using our own measurement, which is based on the official PASCAL VOC mAP implementation [5], we achieve 28% mAP on the validation set. We did this computation on the validation set because the ground truth annotations are not publicly available for the test set.

Regarding speed, on the same mobile device, a method based on MobileNet [21] and SSD [10] achieves around 80 milliseconds (ms) per frame, while our method achieves around 190 ms per frame.

VI. Conclusions and future work

In conclusion, our solution aims to ease the use of public transport by VIPs. The first step that we have taken in doing this is creating an object detection system that can recognize buses, cars, or vehicle registration plates. This part is implemented using a custom version of YOLOv2 [16] and we obtain, on the test set, a mAP of 70.03%, and for the bus class, we obtain an average precision of 90.01%, for the car class 64.04% and for the vehicle registration plate 56.68%, with a speed of around 5 FPS on a mobile device. We have also trained a model on the COCO dataset that achieves around 0.4% mAP on the test dataset. The second part is represented by the mobile application, which serves both as an object detection system visualizer and as a proof of concept for assistive technology for the VIPs that uses object detection. This is illustrated by the accessible live object detection, in which the predictions are not visualized, but converted to sound and played using the mobile device speakers.

There are several parts that can be improved, such that the proposed method attains state of the art results both in terms of accuracy and computational complexity, and we leave them as future work. Firstly, the dataset could be enhanced with images with bad lights or weather, or night images. Recent advances have shown that data-centric AI yields better results than model-centric AI, therefore the dataset could use more attention, in the sense that bad ground truth annotations should be found and fixed. Other techniques presented in the other YOLO papers such as training with images of different sizes, could prove useful.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv*, abs/2004.10934, 2020.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[3] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *ArXiv*, abs/1708.04552, 2017.

[4] Lamya El Alamy, Sara Lhaddad, Soukaina Maalal, Yasmine Taybi, and Yassine Salih-Alj. Bus identification system for visually impaired person. In *2012 Sixth International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 13–17, 2012.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[7] Saeed Khazaee, Ali Tourani, Sajjad Soroori, Asadollah Shahbahrami, and Ching Y. Suen. A Real-Time License Plate Detection Method Using a Deep Learning Approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12068 LNCS, pages 425–438, 2020.

[8] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. OpenImages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[9] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, and et al. The Open Images Dataset V4. *International Journal of Computer Vision*, 128(7):1956–1981, Mar 2020.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[11] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations*, 2017.

[12] Wai-Ying Low, Mengqiu Cao, Jonas De Vos, and Robin Hickman. The journey experience of visually impaired people on public transport in london. *Transport Policy*, 97:137–148, 2020.

[13] Anh Huynh Ngoc. YOLOv2 implementation. Accessed: 25.03.2022, https://github.com/experiencor/keras-yolo2, 2019.

[14] Geneva: World Health Organization. World report on vision. *World Health Organization Publications*, page 77, 2019.

[15] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[16] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[17] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[18] RenewSenses. Travis, the ultimate personal assistant in a discreet add-on to you smartphone. Accessed: 27.03.2022, https://renewsenses.com/, 2021.

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *MICCAI 2015. Lecture Notes in Computer Science*, 2015.

[20] Mohammad Amin Sadeghi and David Forsyth. 30hz object detection with dpm v5. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 65–79, Cham, 2014. Springer International Publishing.

[21] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[22] Google Brain Team. Integrate object detectors. Accessed: 20.06.2022, https://www.tensorflow.org/lite/inference_with_metadata/task_library/object_detector, 2022.

[23] Angelo Vittorio. Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset. Accessed: 25.03.2022, https://github.com/EscVM/OIDv4_ToolKit, 2018.

[24] Tsung-Yi LinMichael MaireSerge BelongieJames HaysPietro PeronaDeva RamananPiotr DollárC. Lawrence Zitnick. Microsoft coco: Common objects in context. *European conference on computer vision (ECCV)*, pages 740–755, 2014.