ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Sistemi Digitali M

COPY-PASTE DATA AUGMENTATION FOR DOMAIN TRANSFER ON TRAFFIC SIGNS

CANDIDATE

Pierpasquale Colagrande

SUPERVISOR

Prof. Stefano Mattoccia

CO-SUPERVISORS

Dr. Maheen Rashid-Engström

Eng. Marco Rovinelli

Academic year 2021-2022

Session 3rd

Contents

Abstract

City streets carry a lot of information that can be exploited to improve the quality of the services the citizens receive. For example, autonomous vehicles need to act accordingly to all the element that are nearby the vehicle itself, like pedestrians, traffic signs and other vehicles. It is also possible to use such information for smart city applications, for example to predict and analyze the traffic or pedestrian flows.

Among all the objects that it is possible to find in a street, traffic signs are very important because of the information they carry. This information can in fact be exploited both for autonomous driving and for smart city applications. Deep learning and, more generally, machine learning models however need huge quantities to learn. Even though modern models are very good at generalizing, the more samples the model has, the better it can generalize between different samples.

Creating these datasets organically, namely with real pictures, is a very tedious task because of the wide variety of signs available in the whole world and especially because of all the possible light, orientation conditions and conditions in general in which they can appear. In addition to that, it may not be easy to collect enough samples for all the possible traffic signs available, cause some of them may be very rare to find.

Instead of collecting pictures manually, it is possible to exploit data augmentation techniques to create synthetic datasets containing the signs that are needed. Creating this data synthetically allows to control the distribution and the conditions of the signs in the datasets, improving the quality and quantity

of training data that is going to be used. This thesis work is about using copy-paste data augmentation to create synthetic data for the traffic sign recognition task.

Chapter 1

Introduction

1.1 Data-hungry machine learning

Modern machine learning techniques like Deep Learning are notoriously data hungry: this means that in order for these systems to reach high performances, they need to be trained with huge quantities of data. In fact, these systems need to work in test environments in which there is no standard condition, so they may encounter data that is very different from the type of data they have been shown during the training phase. In order thus for these systems to be robust to the majority of testing conditions, they need to be shown, during the training phase, all the possible conditions in which data can appear in the testing environment. This is of course impossible: even if humans had the capabilities to collect and store data in all the possible conditions, there will still be outlier situations, rare conditions or conditions that were not taken into account when the dataset was built. Thankfully, deep learning and, more generally, modern machine learning models do not need to be shown data in all the possible situations because they have generalization capabilities: their design and their learning nature allows them to cover a larger number of testing conditions with respect to the ones displayed during the training. However, the harder the task, the more difficult is to collect high-quality data for these systems to learn and generalize well. So, for extension, the lower the quality

of the data, the lower the generalization capabilities of these systems is.

Therefore, in order to improve the generalization ability and, by extension, the performances of these systems, there are three main possibilities:

- improve the model architecture: this solution requires a huge human effort, because experts in the domain need to study and design new models that better suit the desired task in order to improve the generalization abilities of the model; for example, Khan et al. made a comparison study between different architectures in the task of image classification on different datasets [1]

- improve the quality and quantity of the data: this solution also requires a huge human effort, because the data that is collected needs to be annotated, which is a time consuming task, needs to cover a huge amount of possible testing conditions and in general needs to be high in quantity and have high quality;

- improve the training process: the generalization of the model can be improved by using techniques like dropout regularization [2], batch normalization [3], hyperparameter tuning [4], transfer learning [5] and pretraining [6]

All these techniques also help with preventing the overfitting problem, namely the problem of a machine learning problem to stick too much to the training data such that it becomes unable to work properly in a testing environment.

If we have very bad data, no amount of modeling will help us since we still do not have models that reach very high performances with very bad data. On the other hand, with an awful model it is not possible to do anything, for example applying a linear classifier to the raw pixels of an image to classify it is not enough. This means that depending on the starting point for model and data, it would make sense to improve either the model or the data, or maybe

both of them. For example, if we start from a SVM model, it makes sense to improve the model itself. On the other hand, while improved models are publicly available and are often suited for many domains, data for a specific task is not publicly available in quality and quantity.

In addition to that, models that are too big will require more data in general and so in that situation it might be best to reduce the model size if it is not feasible to reach huge amounts of data, so depending on the situation we might want to change the model or the data.

We thus need a way to improve our training data without requiring too much human effort while doing this. The solution to this is a technique called data augmentation.

1.2 Object detection

Object detection consist in localization and classification of objects in images.

One of the first commercial application of machine learning to the object detection task was the Viola-Jones object detection framework [7], developed by Paul Viola and Michael Jones. Their system was using Haar features and a learning-based approach to detect multiple object classes. Other approaches used SIFT features [8] or HOG features [9] combined with a machine learning algorithm.

Since the advent of deep learning, many networks have been designed and developed to solve object detection task. As in other computer vision tasks, deep learning proved to be very effective and more powerful with respect to non-deep approaches. The training of such networks is done by feeding them datasets that consist in images carefully annotated with bounding boxes, which are boxes that are drawn around each instance in the image that define where in the image the instance is and how big it is. The network then digests the patches surrounded by the bounding boxes and learns to find similar instances of objects in the images. The output of the inference phase is a bounding box

surrounding an area in the input image, so the network learns to draw bounding boxes around the objects in the images by using bounding boxes in the training set, however it also learns to classify the instances that it detects into the object classes to which they belong and to do this, a class label is assigned to each instance in the training set. These network thus have generally two heads (a head is an output layer producing a certain output), one head producing the bounding box output, the output of the detection phase, and one head producing the classification output, the output of the classification phase.

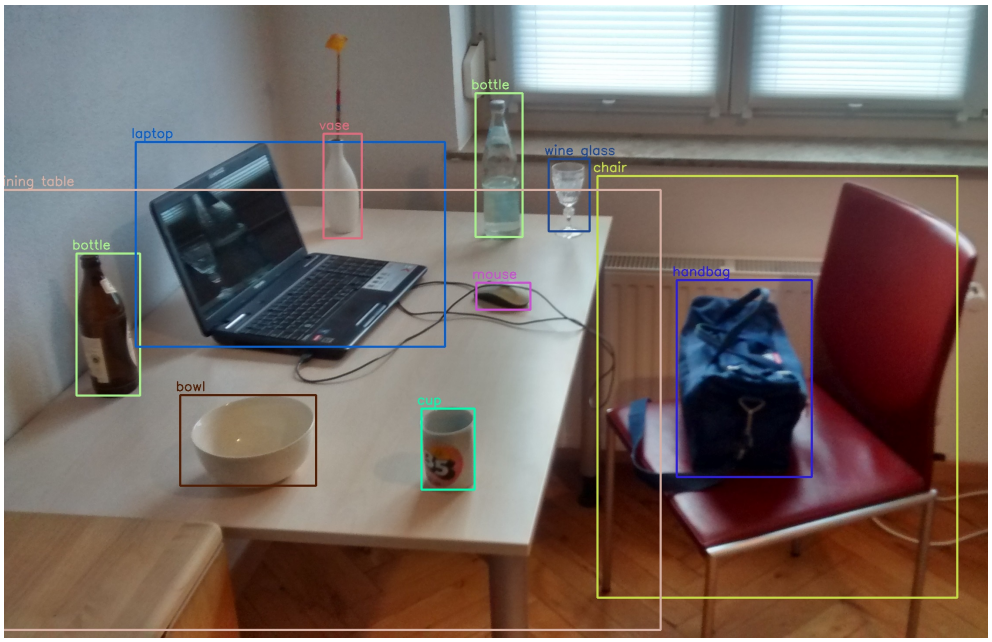


Figure 1.1: Object detection task using YOLOv3 [10] network trained on COCO [11]. Picture from Wikipedia [12].

Lots of datasets for object detection are available, some of them have also been already mentioned in the previous sections. Famous object detection datasets are COCO [11], KITTI [13] and Pascal [14].

On the other hand, among the most successful object detection networks there are networks based on Region Proposal mechanisms like R-CNN [15], Fast R-CNN [16] and Faster R-CNN [17]. While having some differences, all these networks share the same core idea of a mechanism that proposes Region of Interest inside the input image, namely regions that can contain an object.

Moreover, all these R-CNN variants share the same two-stage detection idea: the network can be divided into two main stages, the stage that runs an expensive backbone once per image to produce region proposals and a second stage that is executed once per proposal area that produces the bounding box and the classification output for each proposal area (the Regions of Interest proposed by the first stage of the network).

A further improvement in detectors was made with one-stage detectors that, differently from two-stage detectors, removed the second stage and expanded the first stage to produce also bounding boxes and classifications. Very famous examples of one-stage detectors are the SSD network [18], the FCOS network [19] and the very famous YOLO network, which has now come to its fifth version [20, 21, 10, 22, 23]. Other famous detectors are RetinaNet [24] and EfficientDet [25]. The research in this field is still going on and new networks are being developed every year, allowing to reach higher performances every time, being this indicative of the fact that while data augmentation is a key part of the entire learning procedure, the architecture of the network and improvements in this architecture also make the difference.

Model	**Training set**	**VOC 2007**	**FPS**
R-CNN [15]	07	66.0 %	-
Fast R-CNN [16]	07	66.9 %	-
Fast R-CNN [16]	07+12	70.0 %	0.5
Faster R-CNN [17]	07+12	73.2 %	7
SSD 300 [18]	07+12	74.3 %	46
SSD 512 [18]	07+12	76.8 %	19
YOLOv1 [20]	07+12	63.4 %	45
YOLOv2 288 × 288 [21]	07+12	69.0 %	91
YOLOv2 352 x 352 [21]	07+12	73.7 %	81
YOLOv2 416 × 416 [21]	07+12	76.8 %	67
YOLOv2 480 × 480 [21]	07+12	77.8 %	59
YOLOv2 544 x 544 [21]	07+12	78.6 %	40

Table 1.1: Performance comparison between different architectures on the object detection task on Pascal VOC 2007 test set. In the training set colum, 07 means that the training set used is the VOC 2007 trainval set, 12 means VOC 2012 trainval and 07+12 means the union of both of them. Results are given in terms of mAP percentage for the two validation set and FPS over VOC 2007.

From Table 1.1, it is possible to see how each architecture improves the mAP score, while also improving the frames per seconds that the architecture can process. We can see that in some cases, like with YOLOv1, the performance compared to, for example, Faster R-CNN, is lower in terms of mAP but much higher in term of FPS. Moreover, we can see that architectures that are bigger in scale improve the mAP scores at the expense of FPS. This is indicative of the fact that there must be a trade-off between the mAP score, namely the performance of the architecture in the object detection task, and the FPS at which the architecture operates. Depending on the device on which we want to deploy our object detection system and on the goal we would like to accomplish, we may choose an architecture that reaches a lower mAP but higher FPS or viceversa.

For example, another class of networks called MobileNet [26] was developed by Google for mobile applications. This network is generally suited for deployment on smartphones and mobile devices which do not have an hardware that is as powerful as the non-mobile counterpart, meaning that they cannot run more advanced networks at real-time performances. Moreover, this network trades slightly lower mAP scores for real-time inference on mobile devices. As it is possible to see from table 1.2, MobileNet has a slightly

Framework Resolution	**Model**	**mAP**	**Billion Mult-Adds**	**Million Parameters**
SSD 300	VGG	21.1 %	34.9	33.1
	Inception V2	22.0 %	3.8	13.7
	MobileNet	19.3 %	1.2	6.8
Faster-RCNN 300	VGG	22.9 %	64.3	138.5
	Inception V2	15.4 %	118.2	13.3
	MobileNet	16.4 %	25.2	6.1
Faster-RCNN 600	VGG	25.7 %	149.6	138.5
	Inception V2	21.9 %	129.6	13.3
	MobileNet	19.8 %	30.5	6.1

Table 1.2: COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95). Table 13 from [26].

lower mAP score but a much lower number of parameters compared to bigger architectures like VGG and Inception V2 in the task of object detection on COCO.

More studies and comparisons can be found on the papers of the various architectures, but these studies give us the idea that in general architectures are also important, depending on the situation in which we are and on what we want to achieve.

1.3 Traffic sign recognition

The task of traffic sign recognition consists in an object detection task in which the objects to detect and classify are traffic signs. In this task, there are various challenges to deal with:

- heterogeneous conditions: as with general object detection, the conditions in which traffic signs appears are very different; in a real environment, we may have differences in brightness, contrast, noise, orientation and position with respect to the camera (and thus scale in the image), saturation and many more; this is an important thing to deal with since the absence of representation of some of them in the training set may negatively influence the performances of the detector

- different signs: signs have different colors, shapes and icons that are associated to different meanings and the detector must be able to recognize all of them and distinguish them; some signs may appear very similar because of their shape and color while meaning two different things, the detector must be able to deal with these differences

- country differences: each country has its signs with differences in colours, shapes and icons; for example, Swedish traffic signs have a yellow background, while Italian traffic signs have a white background; moreover, even when the signs are pretty similar (e.g. Italian and Swiss signs

are very similar in color, shape and icons), they may still have some slight differences that may harm the robustness of the detector

To summarize, traffic signs may be very different from situation to situation, depending on the country in which we operate, the context and the environment. In order to make our detector robust to all these differences, we must train in a way that allows to reach high generalization capabilities and to do this, as we saw before, we can either work on the architecture or work on the training set, or we can do both things. In Picture 1.2 we can see various traffic signs in different conditions and states.

Figure 1.2: Different traffic signs in various conditions. Picture from [27].

1.4 Data augmentation

Data augmentation is a technique, now widely used in deep learning, to improve the training data by either improving existing data or by creating synthetic training samples. Data augmentation was introduced in deep learning applied to computer vision [28] in the first place and was exploited to increase sufficiency and diversity of training sets by creating new training images starting from the real ones, however it quickly moved to other domains, like natural

language processing because of its effectiveness [29]. The very first application of data augmentation on Image Classification tasks can be found in LeNet [28], where data augmentation was introduced in the form of data warping.

1.4.1 Basic image augmentations

The very first application of data augmentation as we know it today was done in AlexNet [30], where the images used to train the network were augmented by cropping 224 x 224 patches from ImageNet, by applying to those patches random horizontal flips and by changing the intensity of the RGB channels using PCA color augmentation.

The most simple image augmentation techniques are thus the ones involving basic image manipulation techniques in which new samples are generated by changing the geometric features (e.g. shape, size, orientation with respect to the camera etc.) and photometric features (e.g. brightness, contrast, hue, saturation etc.) of the training images. Additional augmentations that can be in some way included in the photometric ones are the quality deficit augmentations that aim to reproduce synthetically the quality deficit characteristic that digital images have like noise, blur etc.

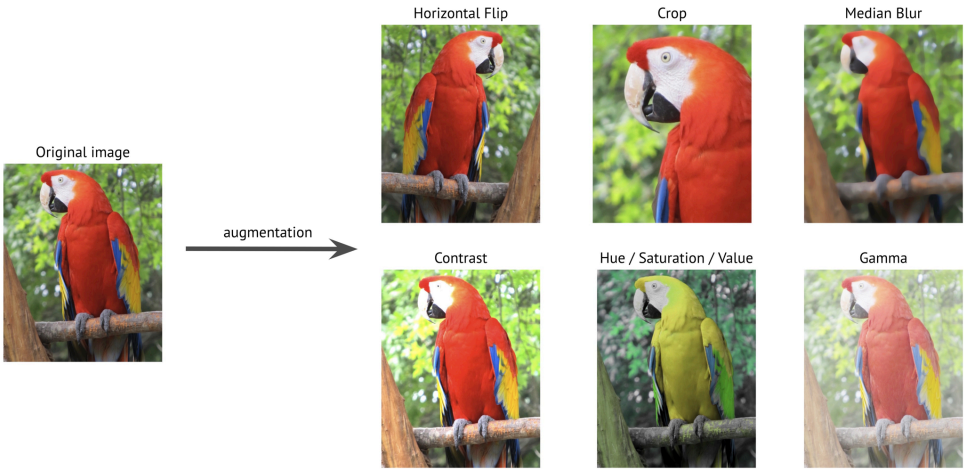


Figure 1.3: Some basic geometric, photometric and quality deficit augmentations. Picture from [31].

These kind of augmentations should affect the image appearance while keeping the label consistent with the content of the image, but this may not be a standard behavior. Depending on the dataset, the type of transformation applied and the degree of this transformation, the label after the augmentation phase may not be preserved anymore. For example, if we consider a dataset like ImageNet, there is a very low possibility of invalidating a label by applying an augmentation. If we take for example the image of a cat from ImageNet, we rotate it, flip it and change the brightness and contrast of the image, in the end the image will still be an image of a cat, as long as the augmentation is reasonable and does not alter completely the picture by, for example, turning it to a picture that is completely black. On the other hand, if we take a digit image from MNIST dataset, there is a very high possibility that the application of certain image augmentation techniques will invalidate the label. For example, by taking an image representing the digit ”6”, rotating it by 180 degrees will turn the image from a picture representing a ”6” to a picture representing a ”9”, invalidating the label. In that case, we need to change the label of the augmented image accordingly in order to make the sample valid. However, if we flip the image of the 6 vertically, it will not represent a number at all, so we cannot even correct the label because there will not be a valid corresponding label. This means that depending on the dataset we are using, we should also choose the augmentation type and amount accordingly, in order to avoid these kind of problems.

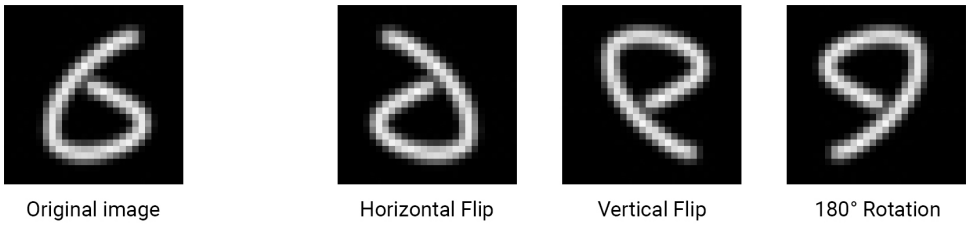


Figure 1.4: Label invalidation of MNIST ”6” digit with standard geometric augmentations.

Geometric augmentations

Geometric augmentations consist in generating new training samples starting from the available ones by changing the geometric features of these. The most used geometric augmentations are:

- Rotation: consists in rotating the image right or left by a certain degree amount and cropping the image accordingly
- Flipping: consists in flipping the image with respect to one of the two axes
- Cropping: consists in cropping the image in order to extract and use only a patch of the original image
- Translation: consists in shifting the images left, right, up or down while filling the remaining space with either noise, with a fixed color or with the closest pixel color
- Scale: consists in scaling the image, changing its original size, while filling the remaining space with either noise, with a fixed color or with the closest pixel color, in order to keep the actual dimension of the images the same
- Shearing: consists in shearing the image along one axis while filling the remaining space with either noise, with a fixed color or with the closest pixel color

Geometric augmentations also include any other type of perspective or affine transformation that modifies the original image. At the same time, the remaining space is filled with either noise, with a fixed color or with the color of the closest pixel.

Photometric augmentations

Photometric augmentations, instead, consist in generating new training samples, starting from the available ones, by changing their photometric features of these. The most used photometric augmentations are:

- Brightness: consists in changing the brightness of the image
- Contrast: consists in changing the contrast of the image
- Saturation: consists in changing the saturation of the image
- Hue: consists in changing the hue of the image

Photometric augmentations also include any other modification to the color channels, such as channel isolation, histogram equalization, exposure etc.

Other photometric augmentations are the ones that aim at generating new training samples, starting from the available ones, by degrading the quality of these images. These include:

- Noise injection: consists in injecting any kind of noise (Gaussian, speckle, salt & pepper, Poisson etc.) in the image
- Blur: consists in applying any kind of blur (Gaussian, motion blur etc.) to the image
- Sharpen: consists in applying any kind of kernel that sharpens the image

1.4.2 Image masking augmentations

Another very common augmentation technique consists in creating new training samples, starting from real ones, by occluding or masking parts of the image. In this case, we are applying some kind of dropout regularization directly on the image, by masking part of it, forcing the network to focus on other areas of the image.

The most common image masking techniques are:

- Random Erasing [32]: consists in removing parts of the image of random size and shape and in random positions and replacing them with noise or with a flat color; random erasing; it is called CutOut [33] when the area of the image that is masked is a square
- Hide and Seek [34]: the image is divided into a grid of NxN patches and each patch has a probability of being masked and is masked according to that probability
- Grid Masking [35]: the image is masked in a grid-like fashion, so similarly to hide and seek, but in this technique the areas that are actually masked follow a grid-like pattern (visual example in Picture 1.5)

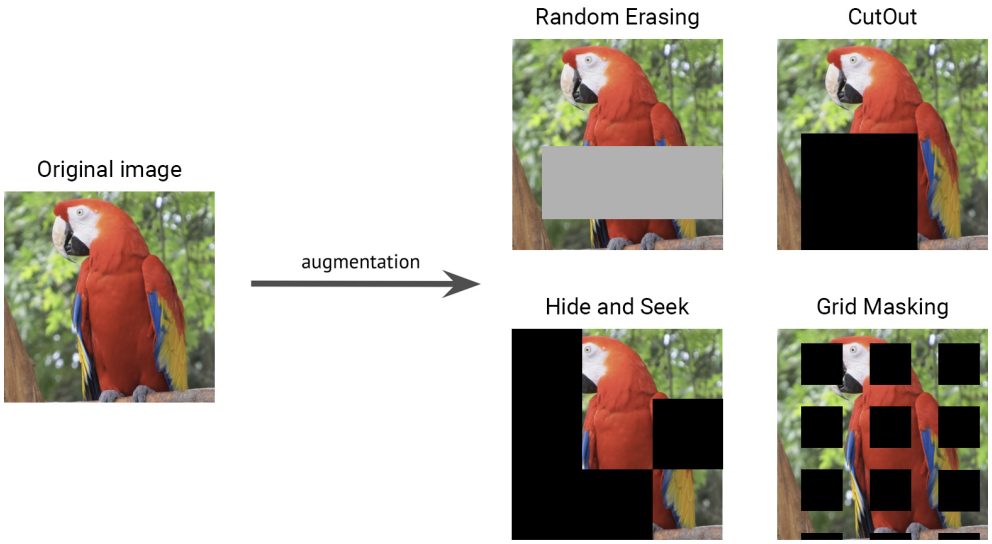


Figure 1.5: Image occluding augmentation techniques.

1.4.3 Image mixing augmentations

Differently from the previously seen basic augmentations, which were augmenting the training set by preserving label consistency, image mixing augmentation aim at creating new training samples by mixing two or more training images using different methods. Moreover, in this case, the label is changed

accordingly to the mixed images. This creates totally new training samples both visually and semantically speaking because, differently from the previous augmentation techniques, the new images will have more than one meaning: for example, if we mix an image of a cat and an image of a dog, accordingly to the percentage of the total image area occupied by the two images, we should also change the label accordingly. If the cat image occupies 80% of the output augmented image and the dog image occupies the remaining 20%, the label of the output augmented image will not be a standard encoding in which the represented class has value 1 while all the others have value 0 (the one-hot encoding), but rather a modified version in which the class cat will have value 0.8 and the class dog will have value 0.2, while all the other classes will have value 0. This way, the label values for all the classes still sum up to one as with one-hot encoding, however a different weight is given to the two classes represented in the image accordingly to the area that the two classes occupy in the image.

The most common image mixing techniques are:

- MixUp [36]: consists in multiple (usually two) training images that are blended together, regulating their opacity level and changing the label accordingly to this opacity level (e.g. a picture of a cat at 80% opacity and a dog at 20% opacity will have 0.8 value in the label for the ”cat” class and 0.2 value for the ”dog” class in the label, while the other classes will have value 0)

- CutMix [37]: similar to CutOut, however instead of removing patches from the images, random patches in the image are replaced with patches of the same size from other images; in this case, the label is changed according to the percentage of the image occupied by the class (e.g. 20% of the image is cat and 80% of the image is dog, so ”cat” class will have 0.2 value and ”dog” class will have 0.8 value in the label while the other classes will have value 0)

- Mosaic [22]: similarly to CutMix, this technique combines 4 training image by taking a single patch from each of these images and by combining them into a new training image; as with CutMix, the label is changed according to the percentage of the image occupied by the class

While not being intuitive at all and while also being the resulting image not useful for a human observer, these techniques have proved to be very effective.

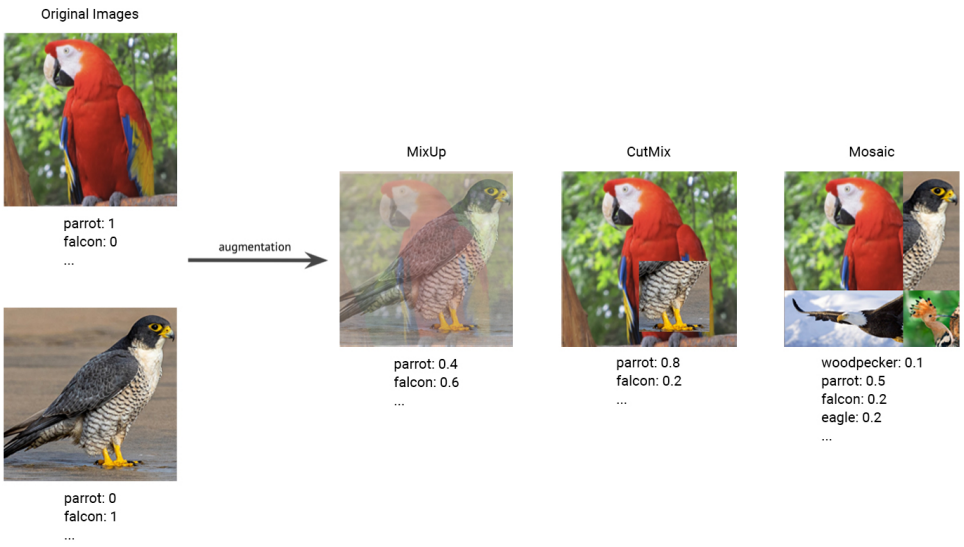


Figure 1.6: Image mixing augmentation techniques.

1.4.4 GAN-based image augmentation

Since the introduction of GANs [38] in 2014, these networks have then been used as an augmentation technique to generate new synthetic training samples to use in various deep learning tasks. For example, in the medical field, GANs have been used to synthesize new training samples for image segmentation [39]. GANs can thus be used in any deep learning tasks to create synthetic data to use for training. Shorten et al. [40] made a very detailed survey on the history and usage of the various data augmentation techniques, including GAN-based augmentation techniques.

1.4.5 Effectiveness of image augmentation

Shorten et al. [40] conducted a survey study on data augmentation for deep learning. Table 1.3, which corresponds to Table 1 of their paper, shows the effectiveness of different basic data augmentation techniques on the task of image classification on Caltech101 dataset. It is possible to see that the usage of each augmentation technique improves the accuracy with respect to the baseline, namely the network without any augmentation. The network architecture is explained in Section 3 of the paper by Taylor et al. [41]. Table 1.3, in fact, also corresponds to Table 3 of [41].

Method	**Top-1 accuracy (%)**	**Top-5 accuracy (%)**
Baseline	48.13 %	64.50 %
Flipping	49.73 %	67.36 %
Rotating	50.80 %	69.41 %
Cropping	61.95 %	79.10 %
Color Jittering	49.57 %	67.18 %
Edge Enhancement	49.29 %	66.49 %
Fancy PCA	49.41 %	67.54 %

Table 1.3: Table 1 from [40] showing the effects of various basic augmentation techniques on image classification task on dataset Caltech101. Results are given in terms of accuracy percentage.

In the same paper, we can find a similar study about the CutOut technique. Table 1.4, which corresponds to Table 2 of [40], shows the effectiveness of CutOut augmentation techniques on top of basic augmentation techniques on the task of image classification on datasets CIFAR10, CIFAR100 and Street View House Numbers datasets. It is possible to see that when using CutOut, the same network reaches a lower error rate.

Moreover, Humza [42] conducted a survey on image mixing and masking augmentations in the task of image classification and object detection. In Table 1.5, which is an extract of Table 2 of [42], it is possible to see how mixing and masking augmentations allow to reach higher mAP in object detection task compared to a baseline method without these augmentations on COCO 17 and VOC 07 datasets, using Faster-RCNN model.

Method	C10	C10+	C100	C100+	SVHN
ResNet-l8	10.63 %	4.72 %	36.68 %	22.46 %	–
+ CutOut	9.31 %	3.99 %	34.98 %	21.96 %	–
WideResNet	6.97 %	3.87 %	26.06 %	18.8 %	1.60 %
+ CutOut	5.54 %	3.08 %	23.94 %	18.41 %	1.30 %
Shake-shake regularization	–	2.86 %	–	15.85 %	–
+ CutOut	–	2.56 %	–	15.2 %	–

Table 1.4: Table 2 from [40] showing the effects of CutOut augmentation on image classification task on CIFAR10, CIFAR100 and Street View House Numbers datasets. Results are given in terms of error percentage. These results also include regular augmentation techniques.

Method	Faster-RCNN backbone	Baseline (mAP)	Augmented (mAP)	Test Set
CutOut	ResNet-50	76.71 %	77.17 %	VOC07
CutMix	ResNet-50	76.71 %	78.31 %	VOC07
MixUp	ResNet-50	76.71 %	77.98 %	VOC07
GridMask	ResNet-50 + FPN	37.4 %	38.3 %	COCO17
Random Erasing	VGG-16	74.8 %	76.2 %	COCO17

Table 1.5: Extract of Table 2 from [42] showing the effects of mixing and masking augmentations on the object detection task on COCO 17 and VOC 07 datasets. Results are given in terms of mAP percentage.

This results prove that the use of data augmentation helps both in the image classification and object detection tasks. As said before, this is mainly due to the fact that increasing the number of training sample and the variance between those training samples helps reducing overfitting, because the network will be shown more and more samples in more and more situations, reducing the distance between the training set and the validation/test sets.

1.5 Copy-paste data augmentation

The augmentation techniques showed in the previous section are now considered a standard for computer vision tasks. These techniques have proved to be extremely effective in a variety of computer vision tasks and for this reason

they are being used widely by researchers to the point that famous frameworks like PyTorch [43] or TensorFlow [44] now include methods in their codebase to provide the possibility of augmenting the training sets when training for any task. Those basic, masking and mixing augmentation techniques are thus considered now a must have for each deep learning research project in order to improve the results of their methods. However, research is also continuing in the field of data augmentation and new augmentation techniques are tested constantly.

A recent augmentation technique is copy-paste augmentation [45]. The goal of this technique is still to produce new training samples to use during training, in order for the network to have more samples and situations to digest during the training, but this technique achieves this objective in a different way: copy-paste augmentation, as the name suggests, copies instances of objects from one image and pastes those instances onto another image. This technique thus works similarly to the previously specified image mixing augmentation techniques, like CutMix [37], MixUp [36] and Mosaic [22], but instead of combining or blending rectangular/square patches from different images, it directly combines two or more images by copying the instances of objects from one or more images to another, which will serve also as a background image.

This technique was introduced by Ghiasi et al. [45] which studied the use of copy-paste augmentation in the task of instance segmentation. They found out that creating new training samples by copying instances from one image to another helped in the task of image segmentation. When copying instance from an image to another, the previously showed augmentation techniques can also be applied at an instance level by changing the position, scale, rotation, brightness, contrast and all the photometric/geometric features of the pasted instance in order to generate more than one training sample from the same pair of original training images. They used this technique to create plenty of new synthetic training samples to use to train a network for image segmentation.

In Figure 1.7 it is possible to see how using two images with their instance segmentation masks allows to create multiple synthetic training samples, by simply copying the instances from an image to another and by varying the instance geometric features as scale, position, orientation etc. Another degree of freedom in this operation is the subset of instances that we are copying from an image to the other. Moreover, we still have photometric and quality deficit augmentations that we can apply to the copied instances, so we have various degrees of freedom at instance-level when applying copy-paste augmentation, but we also have degrees of freedom at an image-level.

Figure 1.7: Copy-paste augmentation allows to obtain multiple synthetic training samples with two original training images. Picture from [45].

As we can see, this is a very strong augmentation technique because, in combination with the standard augmentations, allows us to create a huge quantity of synthetic training samples to use that can also look useful and understandable to a human eye. Moreover the research work from Ghiasi et al. [45] proves that this is very effective for image segmentation: their paper proved that this technique is robust to backbone initialization, to training schedules, to backbone type and image sizes, along with being addictive to large scale jittering. In all the experimental settings, the addition of copy-paste augmentation increased the AP score for both the object detection and instance segmentation tasks on COCO dataset.

The work by Ghiasi et al. [45], apart from proving the effectiveness of

copy-paste augmentation, also proves that paying more attention to data augmentation helps in improving the performances regardless the used architecture (the backbone) and the scale of it, suggesting that exploring and researching more in the data augmentation field may help in improving the performances of already available architectures in many tasks, without the need of designing and developing new fancy and complex architectures or scaling already available ones to dimensions in which the increase in performance will not be worth compared to the computing power that they will require to be trained. Table 1.6 from [45] shows the robustness of copy-paste augmentation to different backbones, showing how this augmentation technique is effective regardless the backbone. More result tables are available in Chapter 4 of [45], showing the robustness of the method to all the previously mentioned situations.

Model	**Box AP**	**Mask AP**
Res-50 FPN (1024)	47.2 %	41.8 %
w/ Copy-Paste	(+1.0) 48.2 %	(+0.6) 42.4 %
Res-101 FPN (1024)	48.4 %	42.8 %
w/ Copy-Paste	(+1.4) 49.8 %	(+0.8) 43.6 %
Res-101 FPN (1280)	49.1 %	43.1 %
w/ Copy-Paste	(+1.2) 50.3 %	(+1.1) 44.2 %
Eff-B7 FPN (640)	48.5 %	42.7 %
w/ Copy-Paste	(+1.5) 50.0 %	(+1.0) 43.7 %
Eff-B7 FPN (1024)	50.8 %	44.7 %
w/ Copy-Paste	(+1.1) 51.9 %	(+0.5) 45.2 %
Eff-B7 FPN (1280)	51.1 %	44.8 %
w/ Copy-Paste	(+1.5) 52.6 %	(+1.1) 45.9 %
Cascade Eff-B7 FPN (1280)	52.9 %	45.6 %
w/ Copy-Paste	(+1.1) 54.0 %	(+0.7) 46.3 %

Table 1.6: Table 1 from [45] showing the effects of Copy-Paste augmentation on the task of object detection and instance segmentation on COCO dataset. Results are given in terms of AP percentage.

Copy-paste data augmentation is also useful to re-balance unbalanced datasets. In case of underrepresented classes, in fact, it is possible to generate new training samples for those classes in order to re-balance the datasets or create new

data that appears in situations that are rarely appearing in the dataset.

1.6 Domain transfer

Domain transfer is the task of changing the domain of certain data. For example, regarding traffic sign images, domain transfer consists in changing the traffic signs of one image with other traffic signs, like equivalent traffic signs of another country, so namely by changing Italian traffic signs with Swedish traffic signs or viceversa.

1.7 Thesis assumption

This thesis is about the use of copy-paste augmentation for domain transfer on traffic signs recognition. In this work, we want to prove that copy-paste data augmentation is very useful to solve all the problems mentioned in the previous section, especially the one of domain transfer: since collecting pictures that contain traffic signs is a very tedious and time-consuming task, we can use copy-paste data augmentation to generate new synthetic data that we can use to train an object detection network. Copy-paste augmentation can help us in solving all the three challenges listed in Section 1.2 because we can create new synthetic training samples by pasting traffic signs icons on background images that we already have. Moreover, we can apply standard augmentation techniques to those icons to simulate the different conditions in which the signs can appear, augmenting the pasted signs at an instance level. We can also use many icons such that we can represent all the possible traffic signs we are interested in recognizing and we can also paste icons of traffic signs from different countries to train the system to recognize the differences between various countries. By changing all these parameters when applying copy-paste augmentation, namely the background image we use, the icons we are pasting and the augmentations we are applying to the pasted icons, we can

create thousands of new synthetic images to use for our training. Moreover, we can build fully synthetic training sets in which we can control the distribution of the various signs and the various conditions, or we can use those degrees of freedom to re-balance some training sets that we already have and that appear unbalanced. On top of that, we can also apply some realism when augmenting the pasted icons such that they are augmented accordingly to the context in which they are being pasted, resulting in images looking more realistic to the human eye but also more context-aware.

Copy-paste thus has potentially a lot of advantages when applied to the domain of traffic signs recognition and can help us in solving all the challenges this tasks has without the need to experiment or come up with new detection architectures. Applying augmentations is in fact much simpler that coming up with new effective and high performing architectures. In Picture 1.8 it is possible to see the process of copy-paste augmentation for traffic sign recognition.

Figure 1.8: Copy-paste augmentation for traffic sign recognition. The original icon of the traffic sign (left image) is then augmented with standard augmentations (middle image) and then pasted on a background image to get a new synthetic training image (right image). In the last image, it is possible to see in the bottom right the pasted augmented icon with its syntethically produced annotation, consisting in the bounding box and the class label

This thesis was developed during a collaboration period at Univrses [46],

a Stockholm-based company working with machine learning. Univrses provided a network developed by them that was left unchanged and that we assume being good enough since it works well on real data. Univrses also provided an annotated dataset for a country and wanted to domain transfer this dataset to another country for which they did not have a dataset for training but had a dataset for testing. The results of our technique will be measured in terms of best achieved validation f1 score averaged per class. Moreover, to avoid measuring noise, multiple runs will be conducted for each experiment with different training seeds and the results will then be averaged per run. More details are available in Chapter 5.

Univrses also provided all the datasets used as a starting point for the various tests, along with the training code and the computing resources.

1.8 Thesis structure

This thesis will be structured as it follows:

1. we will first explore the research literature
   (a) we will explore the research background of architectures and models used for traffic sign recognition
   (b) we will explore the research background of standard augmentation techniques used in the domain of traffic sign recognition
   (c) we will explore the research background of copy-paste augmentation used in the domain of traffic sign recognition
2. we will then show the methods used to implement copy-paste augmentation and domain transfer
3. we will explore the available material, namely the data, the model and the used tools

4. we will illustrate the executed experiments and the achieved results; more particularly, we will show the following experiments and results:
   (a) training with real data and validating with real data
   (b) training with synthetic data and validating with real data
   (c) training with both synthetic and real data and validating with real data
   (d) training with domain-transferred data and validating with real data
   (e) validating with synthetic data while training with real data
   (f) validating with domain-transferred data while training with real data
   (g) validating with synthetic data while training with synthetic data
   (h) validating with domain-transferred data while training with synthetic data
5. finally, we will discuss the obtained results and conclude with some future developments

Chapter 2

Background

2.1 Traffic sign recognition

Regarding the use of improved architectures for traffic sign recognition, for example, Wang et al. [47] tested an improved YOLOv5 architecture for traffic sign detection, consisting in a YOLOv5 architecture with an improved FPN, named AF-FPN, which uses the Adaptive Attention Module (AAM) and Feature Enhancement Module (FEM) to reduce the information loss in the process of feature map generation and enhance the representation ability of the feature pyramid. Moreover, they used a set of default augmentations in combination with a search strategy to find the most useful augmentations. They were reporting that their network was reaching higher performances with respect to a vanilla YOLOv5 architecture in the task of traffic signs recognition on dataset TT100K. A comparison with YOLOv5 and other architectures is available in Table 1 of their paper [47]. Moreover, from Table 2.1, which is Table 2 of [47], it is possible to see how all the techniques used by them improved the scores with respect to a standard YOLOv5s network. This gives us the idea that exploring new architectures may also be useful, even though this is not the main focus of this thesis work. In Picture 2.1 from [47] it is possible to see the task of traffic sign detection and the output produced from the detector.

Liang et al. [48] explored the use of an improved sparse R-CNN for traffic

Figure 2.1: Traffic signs recognition using improved YOLOv5. The images on top are the original images that are fed to the network while the images on bottom represent the results produced by the network (the bounding boxes). Picture from [47].

signs recognition, by adding attention and other modules to the architecture, improving the results compared to other architectures. The comparison of their system with others can be found in Tables 4 and 5 of [48], while in Table 2 it is possible to see an ablation study over the components of their system. They also applied augmentation, which we will discuss in the next section.

Tabernik et al. [49] proposed a slightly improved Mask R-CNN for traffic sign detection and recognition that consisted in adapting the network to the task of traffic sign detection. In Table 2.2 we can see how the network adaptation applied by them improves the baseline Mask R-CNN without adaptation.

These papers show that both model architecture and data augmentation can affect performances, even when starting from good, modern baselines like YoloV5.

2.2 Standard augmentation for traffic sign recognition

The literature also includes experiments about the usage of augmentation to train networks for traffic signs recognition. For example, Park et al. [50] proposed a system to update HD maps on autonomous vehicles. In this system, they trained a YOLOv3 model for real-time detection of many street features, including traffic signs. They used standard augmentation techniques to increase the training samples, working at an image level. The used augmentations were brightness, contrast, translation, rotation, affine transformations, Gaussian blur and random erasing. As shown in Table 8 and 9 of [50], a combination of original and augmented datasets for training increased mAP, recall and F1 scores. They reported an increment of 1.9 % on mAP, 2.4 % on recall and 1.3 % on F1 score when not using grouping, while reporting an increment of 11 % on mAP, 11 % on recall and 16.5 % on F1 score when using grouping, solely with the use of additional augmented data.

On another paperwork, Singh et al. [51] trained a CNN based model for traffic signs recognition. While training this model, they also used data augmentation techniques like shearing, rotation, scaling, flipping, shadowing etc. They also used Gaussian blur and medial blur. They were however not showing any comparison to non-augmentation methods.

Wang et al. [47] also used a sophisticated learning strategy to learn which augmentations are the ones contributing the most and thus to learn which augmentations to apply. The search space included Mosaic, SnapMix, Erasing, CutMix, Mixup and Translate X/Y, for a total of 15 operations. In Table 2.1, it is possible to see the contribute of augmentation on mAP score.

Lian et al. [48] applied augmentations on an image level. These were mainly standard augmentations applied to the entire image to create new training samples, like augmentations to simulate different lightning, weather and noise conditions. They also applied sepia, grayscale, blur, channel dropout,

Method	Model	Params	FLOPs	FPS	mAP
YOLOv5s	14.6M	7.193M	17.9G	105	60.18 %
YOLOv5s + augmentation	16.3M	7.193M	17.9G	105	61.31 %
YOLOv5s + AF-FPN	14.6M	8.039M	17.9G	95	62.67 %
Final	16.3M	8.039M	17.9G	95	65.14 %

Table 2.1: Table 2 from [47]. Results given in term of mAP percentage on TT100k dataset. The final model includes both the use of AF-FPN and data augmentation strategy.

CLAHE, color jitter, glass blur, Gaussian blur, horizontal flip, perspective, rain, random erasing, snow simulation, mosaic, fog, brightness, contrast, gamma, sun flare simulation, shadow simulation and low light simulation. They reported that, when applying image level augmentations, AP increased from 61.3 % to 63.8 % and AP50 increased from 92.7 % to 94.8 % when training a RetinaNet-based model, while AP increased from 70.2 % to 72.4 % and AP50 increased from 94.7 % to 96.1 % when training a model based on Faster R-CNN. These results are illustrated on Table 3 of their paper.

2.3 Copy-paste augmentation for traffic sign recognition

Many experiments have already been conducted on the use of copy-paste augmentation for traffic signs recognition. Different researchers tried different techniques based on pasting traffic signs into background images in order to produce synthetic datasets.

By reiterating copy-pasting multiple times with different templates, it is possible to create many realistic images with different traffic signs.

We can divide these experiments in two major categories, non realistic copy-paste and realistic copy-paste.

In non realistic copy-pasting, the copy-paste operation happens in a non-realistic way, meaning that the traffic signs are pasted without taking into account the background and the context on which the signs are being pasted.

This means that each traffic sign will be mostly pasted on a random position in the background picture and its brightness, contrast, orientation, noise etc. will be changed randomly, without considering the brightness, contrast, noise etc. of the image (and the position in the image) in which the template sign will be pasted. This technique will produce, visually speaking, images that do not look very realistic, because the signs will have different visual features compared to the area in which they are being pasted and they may also appear in positions of the image in which they will never appear in reality. However, in most cases, this is more than enough because the network will still learn the features of the signs and because copy-paste augmentation will anyway create a pretty balanced dataset with many traffic signs in many conditions, but the effectiveness of this random augmentation also depends on how much context around the traffic sign the detector uses.

In realistic copy-paste the signs are pasted and augmented accordingly to the background and context in which they are being pasted. In this case annotated datasets are usually used as a reference for the augmentation: the annotated signs in the datasets are used as a reference for the augmentation, namely the real signs in the real images are used as a reference for realism embedding in the augmentations. In this case, the images will be more realistic visually speaking, because the template signs will be pasted in positions in which they normally would appear and their visual features will match the context in which they are pasted. Again, the usefulness of these realistic augmentations depends on how much context around the sign the network uses. The challenging part is the copy and paste of the style from the image to the template and the realistic selection of the pasting positions and orientations for the traffic sign templates. A good compromise between complexity of implementation and quality of results can be achieved with an algorithmic approach, meaning that for photometric features like brightness and contrast, it is possible to compute the average brightness or contrast of the area in which the sign will be pasted and then applying these computed brightness and contrast levels to

the template. For geometric features like the position of the template in the image, it is possible to ignore certain areas of the image in which usually signs do not appear: for example, the top of the image is usually the sky, while the bottom part is usually the road, so these areas can be ignored because signs rarely appear here. It is also possible to just use the annotated traffic signs of some datasets to find realistic positions in which to paste the sign, for example by replacing the existing signs in the real images or just pasting new signs near to the existing ones. Another approach to find the position can be computing the centroids of the signs cluster in order to find positions in the images in which the signs are present on average. For the orientation, instead, it is possible again to use the annotated traffic signs and their corresponding template icons to compute SIFT [8] features, then by using feature matching and homography estimation it is possible to compute a rotation matrix to apply to the template to paste, in order to give it a realistic rotation based on the rotation of the annotated traffic signs. Moreover, it is possible to use the internal parameters of the camera that has been used to capture the background images, if known, in combination with the real world sizes of the signs to define the position, the distance from the camera (the scale) and the orientation with respect to the camera (the rotation) of the signs. Some works extracting photometric and geometric features algorithmically from annotated traffic signs in order to generate new synthetic traffic signs are [48, 49]. Other sophisticated methods consist in training and using Generative Adversarial Networks like StyleGANs [52] or CycleGANs [53] to transfer the style automatically from the image to the templates [54, 55]. In any case, the core process for realistic augmentation consists in using images of datasets already available as background and using the annotated traffic signs as a reference for the realism of the templates.

In order to generate images with traffic signs, we need two things, traffic sign templates, namely traffic signs icon that will be used for the copy-paste operation, and background images that will be used as a background for the

pasted signs.

Based on the templates, we can define two categories of experiments, the ones using traffic sign templates from real images and the one using traffic sign icons. The first category consists in experiments using datasets of road images in which traffic signs are annotated. In fact, these datasets usually provide annotations for the traffic signs like bounding boxes or segmentation masks, so the extraction of the traffic sign templates is straightforward. Once extracted, the signs can be normalized in geometry and appearance and then augmented to create synthetically generated new signs that will be used for the copy-paste operation. The second category consists in experiments using traffic sign icons taken from the web. In fact, by law, most states need to provide traffic signs icons that are already standardize in appearance and geometry since they are usually vector icons. In any case it is possible to get such icons from the web and use them as a starting point for the augmentation; these icons can be augmented to create synthetically generated new signs that can then be used for the copy-paste augmentation and that will be pasted on the background images.

Another categorization can be defined based on the nature of the used background images. We can use both images from the domain of interest and images from other domains. The former consists in experiments using background images taken from the domain of interest, namely images of streets and roads and any setting in which traffic signs can appear. The latter consists in experiments using background images taken from everywhere but the domain of interest, namely any image that does not represent streets or roads or any setting in which traffic signs can appear. These images can be taken from the web or from some famous datasets like ImageNet. In this case, again, the first method can be more useful depending on how much context around the traffic signs the detector uses.

In any of these cases, the core of the copy-paste augmentation procedure is the same: obtaining traffic sign templates, augmenting them and pasting the

augmented signs on background images. The resulting images will be used to train the detectors.

2.3.1 Non realistic copy-paste

One of the components of the system by Liang et al. [48] was copy-paste data augmentation. In fact, they trained their system using a training set that was augmented both on an image-level, by applying standard augmentations to the whole image as discussed in Section 2.2, and on a box-level. The augmentation technique used was thus copy-paste augmentation: what they were doing to create new training images was taking two training samples from the original dataset and copy-pasting some traffic signs with their annotations from one of the two images to the other, to create a new training sample containing new pasted traffic signs that looked semi-realistic. The copy-pasted signs were replacing the original signs in the background image. Moreover, a geometric transformation was applied to the signs that were being copied from one image to another in order to proportion it to the sign they were replacing. On top of that, a Gaussian blur was applied to the border of the patch in order to better blend it with the background. More details can be found in Section 3.3 of [48]. Moreover, Picture 2.2 represents the result of their augmentation procedure.

In Table 2 and 3 of [48] we can see how both image-level and box-level augmentations improved the score for traffic sign detection on dataset BCTSDB (Beijing Union University Chinese Traffic Sign Detection Benchmark) both for their custom improved sparse R-CNN architecture and for standard architectures like RetinaNet or Faster R-CNN, demonstrating the effectiveness of augmentation and copy-paste augmentation in the task of traffic signs recognition and its robustness to the used architecture. From these tables, we can see that box-level augmentations, when applied to RetinaNet, improved AP score from 63.8 % to 64.1 %, while AP50 improved from 94.8 % to 95.2

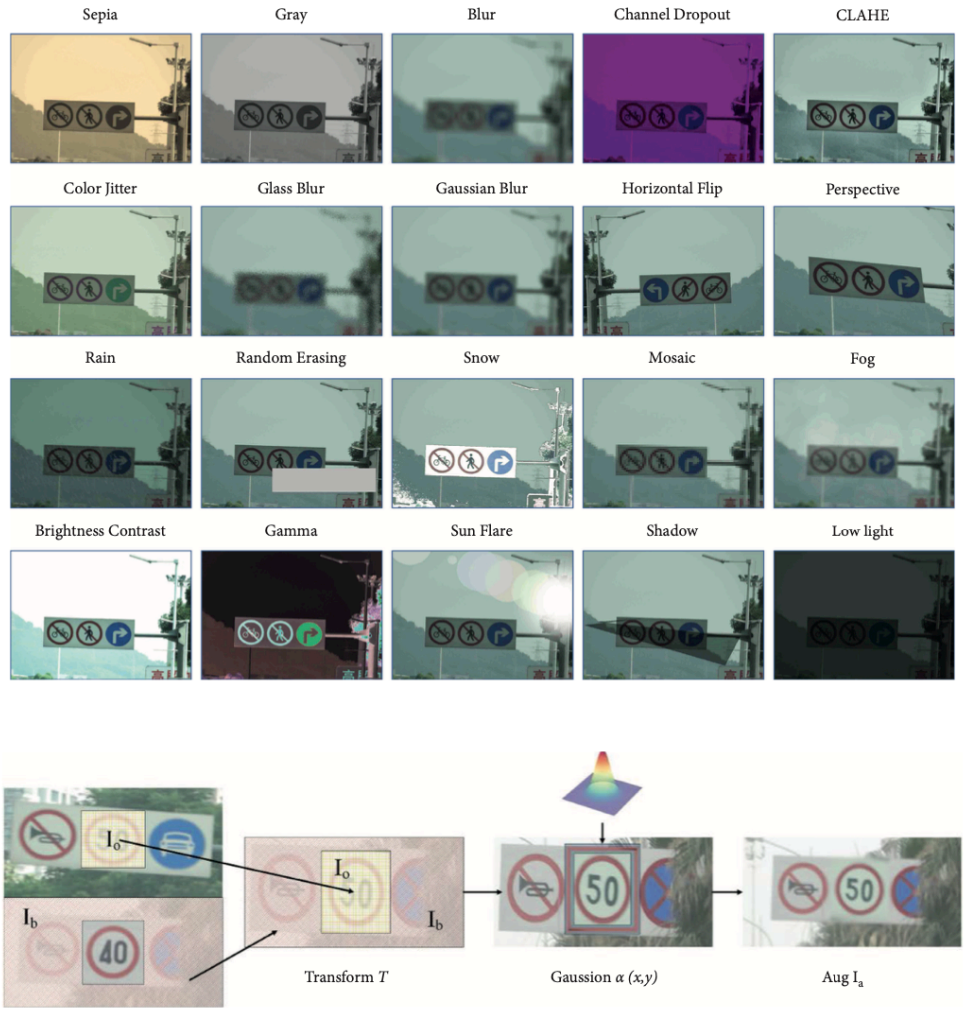


Figure 2.2: Image-level and box-level augmentations used in [48]. In the top, we can see the image-level augmentations, while in the bottom the box-level copy-paste augmentation. Picture from [48].

%. Moreover, when applying box-level augmentations to Faster R-CNN, AP score improved from 72.4 % to 72.7 %, while AP50 improved from 96.1 % to 96.5 %. When applying both image-level and box-level augmentations to their baseline, instead, AP50 improved from 95.8 % to 98.1 %, while AP75 improved from 92.5 % to 94.2 %. In this work, so, the icons or templates, or the signs used for the copy-paste augmentation were directly extracted from the training images and pasted onto other signs in other training images, using their annotations (their bounding boxes) as a reference for the extraction and

replacement. No extra augmentation was applied to the boxes, apart from a geometric transformation to match the dimensions of the sign being copied and the sign being replaced and apart from a Gaussian blur the border of the patch to better blend the patch with the image. This paper thus uses only original training images for the copy-paste augmentation phase. The different lightning, weather and noise conditions were thus addressed not on a box level but rather on an image level by applying regular augmentations on top of copy-paste augmentation to the whole image rather than to the single boxes. Since they were using the bounding box to extract the signs, this means that the patch being extracted will also contain the background of the image, meaning that in the pasting operation we will also paste some background from the source image.

Apart from proposing a slightly modified Mask R-CNN architecture, the core of the work by Tabernik et al. [49] was the usage of copy-paste data augmentation to produce new training samples. In fact, they were extracting traffic sign templates from the training images using their segmentation mask in order to perfectly extract only the sign from the image. The dataset used to extract the traffic sign from was DFG traffic sign dataset. Then, they were normalizing the extracted traffic signs such that they could have a normalized geometry and appearance and finally they were applying transformations and augmentations to the normalized signs in order to produce new signs with different lightning and orientation conditions. As last step, the augmented signs were pasted to street-environment-like background images taken from a subset of the BTS traffic signs dataset, so the background images were actually domain-related images. In Picture 2.3 we can see the result of the generation of synthetic augmented traffic signs performed by [49].

In Table 2.2 we can also see how copy-paste data augmentation improves the performances of the adapted Mask R-CNN network from [49].

In this paper thus, copy-paste augmentation was achieved by using only training images already annotated and by, again, extracting signs from one

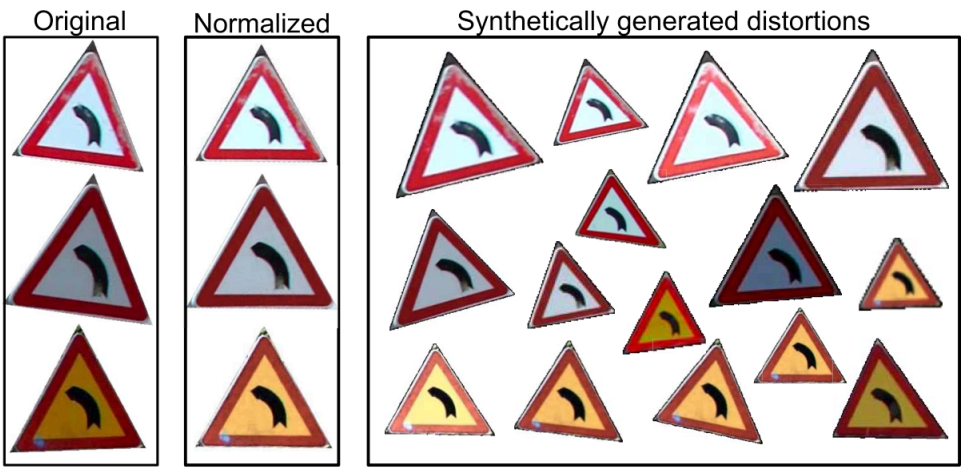


Figure 2.3: Copy-paste data augmentation used in [49]. On the left, we have the original signs extracted from DFG traffic signs dataset. On the center, the same signs normalized in geometry and appearance, while on the right we have the augmented templates generated with synthetic distortions applied to the normalized templates. Picture from [49].

		Mask R-CNN (ResNet-50)	
	No adaptation	**With adaptation**	**With adaptation and augmentation**
mAP 50	93.0 %	95.2 %	95.5 %
mAP 50:95	82.3 %	82.0 %	84.4 %
Max recall	94.6 %	96.5 %	96.5 %

Table 2.2: Results of [49] on traffic sign segmentation on DFG traffic sign dataset. Results are given in percentages. Table 3 from [49].

training image and pasting them onto another training image, like the work by Liang et al. [48] but, differently from the latter one, augmentation is done on a box-level, by augmenting the template (the extracted sign) instead of the whole image. Moreover, templates and background images were not taken from the same dataset. Templates in fact were taken from images of the DFG dataset while the background images are taken from BTS dataset so, instead of using just one dataset for the whole augmentation procedure, two were used. In addition to that, they were using the sign segmentation mask to extract the template sign, so no background was copied from the source image to the target image, differently from the previous work.

Tabelini et al. [56] produced another study about copy-paste augmentation in the domain of traffic signs recognition. What they did was to train a network for traffic signs recognition by using a dataset that was augmented using copy-paste data augmentation. In this case, templates of traffic signs were taken from the internet or from available traffic sign datasets. In fact, country-wise, templates of traffic signs must be available to the public. The used templates were thus icons that were all already normalized in terms of lightning conditions, orientation etc., since they were vector icons. After selecting the icon to paste, this icon was augmented by applying standard augmentations such as brightness and contrast, rotation and geometric transformations and noise. The augmented template was then pasted on a background image by first applying a Gaussian blur to fade the border of the icon and blend it better with the background image. Background images were taken from Microsoft COCO, so the background images were not domain-related. The authors claimed that is better to use non domain-related background images because using background images from the domain of interest is not required and, moreover, using domain-related images may introduce unwanted noise in the training data if these images are not carefully annotated. Images from the domain of interest might eventually present the objects of interest (namely, traffic signs) that if not being annotated will introduce noise in the network prediction. In Picture 2.4 we can see the copy-paste augmentation used by [56], while in Picture 2.5 we can see the images resulting from the augmentation procedure.

In Table 2.3 we can see the results of the application of this augmentation technique. We can see that training the network with a fully synthetic data allowed to reach performances, in terms of mAP, close to the ones of the network trained with real data. On some datasets, moreover, the network trained only with synthetic images was also more performant than the network trained only with real images. However the best performances were achieved when training the network with both synthetic and real data, indicating that augmentation in general is useful when used to augment already existing data,

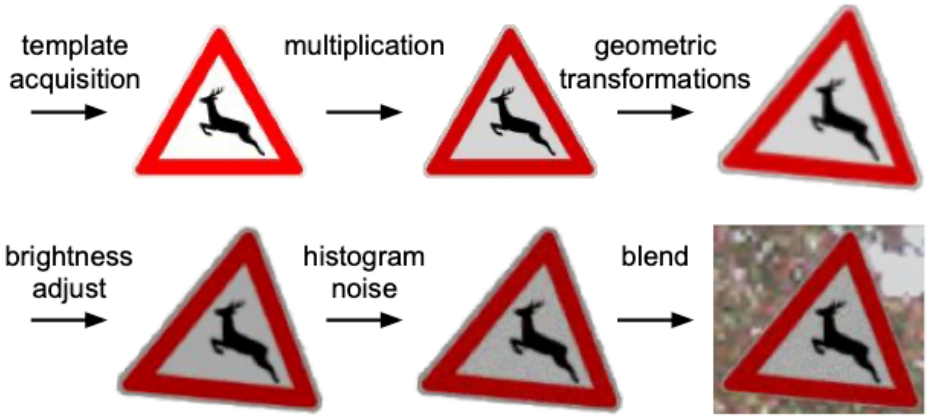


Figure 2.4: Copy-paste data augmentation used in [56]. Picture from [56].

Figure 2.5: Copy-paste augmented images from [56]. Picture from [56].

not when replacing it.

In this work, differently from the previous two, the templates were actually icon taken from the web and not extracted from other training images. Moreover, the used background images were images not from the domain of interest, so images of anything but roads and streets. However, the network trained on this synthetic data proved to be very effective, confirming that copy-paste data augmentation is also useful in the domain of traffic signs recognition. The photometric and geometric augmentations are also applied on a box-level (directly to the templates) instead of an image-level.

In all these three works, the augmentations applied were mostly random,

	mAP		
Training set	**BTSD**	**TT100K**	**GTSD**
Real	85.50 %	89.28 %	79.50 %
Synthetic	80.28 %	83.12 %	91.75 %
Synthetic + real	89.64 %	92.25 %	-

Table 2.3: Results of copy-paste augmentation used in [56] on traffic sign recognition on real data. Table is an extract of Table III from [56].

apart from few of them like the size and position of templates when replacing traffic signs in [48] or the position of the signs in [56]. Everything else as brightness, contrast, noise etc. was randomly chosen, meaning that it was not meeting the context of the position in which the signs were being pasted. However, this technique still proved to be very effective, especially because this way it is possible to control the distributions both of the augmentations and of the pasted signs.

2.3.2 Realistic copy-paste

Horn et al. [54] created a system to replace traffic signs in real world images in a fully automatic way. Their method consisted in using the GTSRB (German Traffic Sign Recognition Benchmark) dataset as a base for the realistic copy-paste augmentation procedure. The copy-paste process consists in two pipelines, the extraction pipeline and the composition pipeline.

The extraction pipeline consists in taking an annotation from the dataset (a real sign patch) and feeding it to a CycleGAN [53] to produce a cartoonized version of the real sign. This cartoon version facilitates the extraction of the binary background segmentation masks, which in turn facilitates the calculation of the traffic sign pose (the homography) by using ORB features [57], feature matching and RANSAC [58] method to estimate the homography by matching features between the cartoonized sign and the straight icon of the corresponding sign (the vector icon of the sign).

Once the homography has been computed, the composition pipeline starts. Starting from a straight traffic sign icon, namely the icon of the sign that we

want to paste, we apply to it the previously computed homography to generate a tilted icon that matches the orientation of the original sign in the dataset. After that, the previously computed segmentation mask is used to extract the cartoonized background from the cartoonized traffic sign. This cartoonized background is then pasted behind the generated tilted icon to produce a reconstructed cartoonized sign with the sign we want to paste. Once the cartoonized version is produced, this reconstructed cartoon sign is fed to the same CycleGAN as before to generate a realistic version of it. The background segmentation mask is used again to extract the real background from the real original sign and to extract the generated realistic sign from the cycled patch. These two are then combined together. During this process, borders are crossfaded to avoid artifacts. The resulting composed imaged is put in place of the original sign patch, producing an augmented realistic image. In Picture 2.6 is is possible to see these two pipelines and the result of the entire procedure from the original sign to the synthetic realistic one.

As a baseline for the experiments, the authors trained a SVM on the GTSRB dataset, namely the dataset containing only real images. Once again, training the same classifier with a mixture of real data and synthetic data, in order to balance the underrepresented classes, increased the accuracy of the classifier from 88.01 % to 89.75 %. More details can be found in Tables III, IV, V, VI and VII of [54], including class specific and sign specific accuracy. This work mainly showed the contribution of their method in the task of traffic sign classification, not detection. From their results, however, it is not clear how much realism contributed with respect to the use of non-realistic copy-paste augmentation.

Konushin et al. [55] tested another approach for realistic copy-paste augmentation in the task of traffic sign recognition. Their method allows for realistic embedding of rare traffic signs classes which are absent in the training set.

The core of their work was to create a functional method to paste traffic

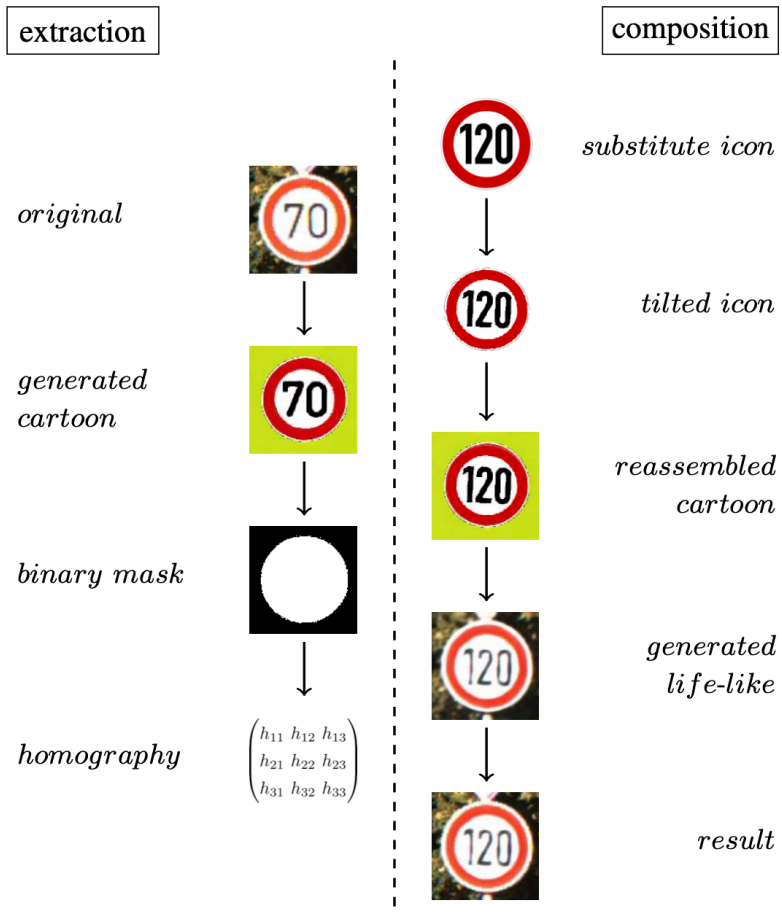


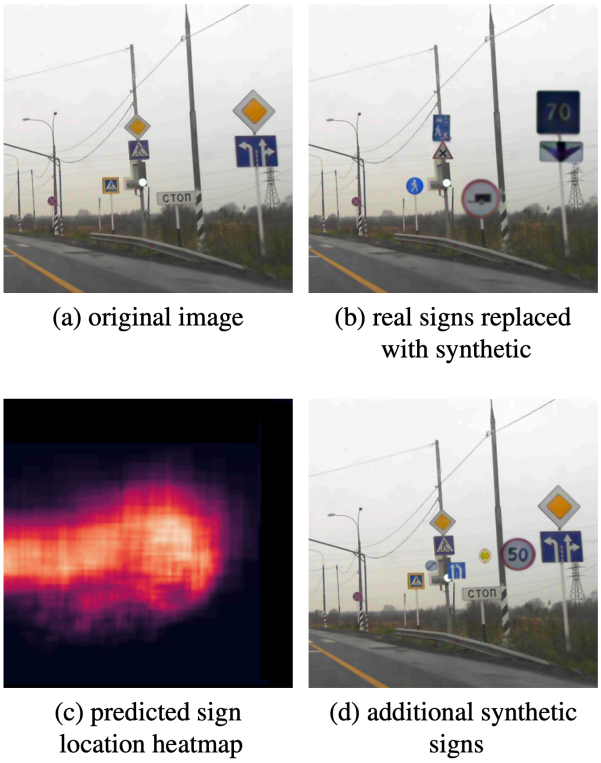
Figure 2.6: Pipelines used in [54] for realistic copy-paste replacement of traffic signs. The resulting patch is substituted to the original patch in the original image. Picture from [54].

signs in real images in a realistic way in order to re-balance the original dataset and embed rare traffic signs. To do this, they explored two approaches, replacement of existing real traffic signs with synthetic ones by inpainting at the place of the real signs, like in [54], and embedding of additional artificial signs in new positions by learning how to find the most suitable position for the new traffic signs.

In both of the cases, a processing of artificial signs is needed in order to make it realistic. To do this, they proposed three methods based on GANs [38], the first two based on CycleGANs [53] and the third based on StyleGANs [52]. In the first approach, called ”pasted”, networks are trained together both for inpainting an processing of embedded traffic sign. In the second approach,

called ”cycled”, the network is similar to the first one but with an additional data stream. In the third approach, called ”styled”, a more advanced generator was used.

These three networks were used to embed the traffic signs. However, authors also trained a neural network that would find appropriate places for additional traffic signs on road images. So, instead of replacing and inpainting the existing real signs, using this network, additional traffic signs were added to the original images and the position in which to paste these signs was computed by this last neural network. From Picture 2.7 we can see the results of their copy-paste augmentation.



(a) original image

(b) real signs replaced with synthetic

(c) predicted sign location heatmap

(d) additional synthetic signs

Figure 2.7: Results of the copy-paste augmentation of [55]. Picture from [55].

To check the effectiveness of their methods, the authors used RTSD (Russian Traffic Sign Dataset) dataset both for the the training, by augmenting the training set of RTSD, and for the testing, by using the testing set of RTSD as

test set. Additionally, they tested both replacement of original signs and pasting of additional signs. They also compared their augmentation method with already existing augmentation methods for traffic signs. These methods were:

- synt: method consisting in sign embedding on the background and random transformation of signs with random parameters to the icon like brightness, contrast, gaussian blur, motion blur etc., namely what we called before non-realistic copy-paste augmentation
- cgi: training images augmented by rendering three-dimensoinal models of traffic signs on pillars in real road images
- cgi-gan: using CycleGAN on top of cgi method to improve the realism of the cgi pictures
- inpaint: synthetic data is created by simply drawing an icon of a traffic sign in the image wihtout any processing

From Table 9 and 10 of [55], we can see how, generally speaking, the use of augmented realistic data improves a little bit the detection performances compared to the use of augmented data without realism. From the tables it is also possible to see how copy-paste augmentation helps compared to using raw data, especially in case of rare classes.

Soufi et al. [59] made another system based on GAN to create copy-paste augmented data in the domain of traffic sign recognition. They were using a GAN-based network called pix2pix to apply realism to copy-pasted traffic sign icons. This work is interesting because it proves that, compared to non-realistic augmentation, realism can also be non useful. From Tables I and II of [59], we can see how the use of their GAN-based realistic augmentation surely helps compared to not having copy-paste augmentation at all, but we can also see how the performances of a classifier trained on data augmented with their technique is pretty close, if not a little worse in some cases, than

the performances of a classifier trained with copy-paste augmentation without realism but instead with standard random non-realistic augmentations. In Figure 2.8 we can see some results of their realistic copy-paste augmentation.

Figure 2.8: Results of the copy-paste augmentation of [59]. Picture from [59].

What we can see from these works is thus that realism may and may not help. Although some of the resulting images from these systems are exceptionally realistic to a human eye, they may not help that much when it comes to traffic signs recognition. Moreover, these GAN-based systems are very complicated to implement and especially to train; in fact, GANs are known to be highly unstable when being trained. So it may not be necessary such an effort to embed some realism into the augmented images, it may be possible to obtain similar results by embedding realism in an algorithmical way. The effectiveness of realism of course, as said before, also depends on how much context around the sign the detector uses or may depend on the technique used to embed realism itself.

Chapter 3

Methods

The first step of copy-paste augmentation was to implement the copy-paste augmentation operation in a non-realistic way. In doing this, we had various degrees of freedom, namely we had to decide what augmentations and how to implement them. In order to make experiments easier, we decided to use the hydra framework [60] to create YAML configurations file for the augmentation procedure. These configuration files are very handy in order to decide the various parameters of the augmentations and what kind of augmentations to apply.

3.1 Copy-paste augmentation

The various augmentations that we implemented for the traffic signs to paste include photometric and geometric augmentations, as well as quality deficit augmentations like noise or blur. We implemented a non-realistic version of each of these augmentations, that modifies these features randomly according to certain parameters defined for each one of these box-level augmentations in the configuration file, and a realistic version which applies the augmentations mainly based on features that we can extract programmatically from the background images.

In Figure 3.1 is is possible to see the difference between our copy-paste

Without realism

With realism

Figure 3.1: Qualitative results of our copy-paste augmentation without realism (top) and with realism (bottom).

augmentation without and with realism. As it is possible to see, the realistic augmentation meets the context in which the sign is pasted. However, as it is possible to see, in the realistically augmented image some signs are nearly transparent or barely visible. The solution to this problem will be described in the sections related to realistic augmentations.

3.1.1 Signs to paste

In order to choose the signs to paste, we defined a parameter in the augmentation configuration that allows us to define the probability that each class of signs will appear in the augmented image. This parameter thus controls the probability distribution of the sign classes in the augmented dataset. By default, each class of signs has the same probability of appearing in an image, but we can use this parameter in the configuration file to define custom probabilites per each sign class. When doing this, the probabilities of the remaining classes are changed accordingly to the ones customly defined such that all the probabilities sum up to 1. This way, we are controlling the distribution of the synthetic signs in the augmented dataset, such that we can re-balance a dataset in case of poorly represented sign classes.

3.1.2 Number of signs per image

Non-realistic

The first parameter of the augmentation procedure consists in the number of traffic signs per image. We need a parameter to decide how many traffic signs per image to paste. Instead of implementing this parameter as a simple number that defines the number of pasted traffic signs per image, we decided to implement this as a foreground/background ratio threshold: before pasting a sign, we compute the ratio between the number of foreground pixels (the number of pixels in the augmented image covered by a traffic sign icon) and the number of background pixels (the number of pixels in the augmented image covered by everything but traffic signs icon). In computing this ratio, we also consider the pixels occupied by the traffic signs that were already annotated in the image, if any. After computing this ratio, we check if this is lower or equal to the threshold defined in the augmentation configuration file and if yes, we paste a sign and repeat the operation until the ratio exceeds the threshold. Of course this method does not define a precise number of traffic signs per

image because, accordingly to the size of the pasted traffic signs,to the number and size of traffic signs already annotated in the images and to the size of the image itself we will end up with a different number of traffic signs per image, however a foreground/background ratio is more useful to control the space occupied by negatives in the image. The foreground/background ratio is configurable in the augmentation configuration file.

Realistic

No realistic version of this augmentation has been implemented because we want to control the percentage of occupied area precisely and we have no need to apply realism to this parameter.

3.1.3 Position

Non-realistic

When pasting the signs, we also have to decide where in the image to paste the templates. The position in which to paste signs is chosen randomly at each pasting operation. In doing this, the major challenge is avoiding overlaps between traffic signs. In order to avoid them, thus, while pasting signs, we keep track of the areas occupied by the traffic signs and we select a position in the image randomly but if the area in which the sign is going to be pasted is already occupied by a sing, we repeat the operation until a free position is selected. In addition to this, a special parameter that controls the padding around the already pasted icons has been used. This parameter defines how many pixels around the already pasted signs we must consider as occupied when selecting a random position for a new sign. This is useful to give a little bit of white space between the signs and to avoid signs being pasted precisely one next to the other. In addition to this, at each pasting operation, we also ignored as much rows of pixels as the height of the sign on the bottom part of the image and as much columns of pixels as the width of the sign on the

right part, in order to avoid signs being pasted fully or half outside the image. For example, if a sign has width 10 pixels and height 15 pixels, we ignore 15 rows of pixels in the bottom of the image and 10 columns of pixels in the left of the image. The only parameter that is configurable in the configuration file for positioning is this padding value around the pasted signs.

Realistic

The position was kept random for all the signs so no realistic positioning was implemented.

3.1.4 Scale

Non-realistic

Before pasting the signs, we scaled them randomly according to a minimum and maximum scale size defined on the augmentation configuration in order to introduce variance in the scale of the pasted signs.

Realistic

No realistic version of scaling was implemented because, again, we want to fully control the scale of the pasted signs since the dimension of the pasted signs also affects the number of pasted signs according to the foreground/background ratio.

3.1.5 Rotation along the three axes

Non-realistic

The signs are also augmented by applying a perspective rotation along the three axes. The template is in fact rotated among the three axes according to a random degree of rotation, that can be either positive or negative. The rotation is implemented as a perspective transformation. The minimum and

maximum degree of rotation per axes is a parameter that can be controlled in the configuration file. Moreover we also decided to add another parameter for each axes which is the probability distribution of the various degrees of rotation. We used four possible values:

- uniform probability distribution: each degree of rotation in the range defined in the augmentation configuration file has the same probability of being selected
- low probability distribution: consists in a Gaussian probability with mean 0 and a very low sigma; this will produce the effect that most of the signs will not be rotated along the current axis or will have a very low degree of rotation, cause the Gaussian will give a high probability to the degrees around 0
- medium probability distribution: consists in a Gaussian probability with mean 0 and a medium sigma; this gives a little higher chance to degrees higher than 0 to be selected, in order to keep a high probability to the degrees 0 but also include higher rotations more often
- high probability distribution: consists in a Gaussian probability with mean 0 and a very high sigma; while still being a Gaussian with mean on 0, this distribution is closer to a uniform one giving more probability of being selected to high degrees of rotation, still giving however a higher probability to rotation degrees in the range the closer they are to 0

With this probability distribution value, which is also another parameter that can be changed in the configuration file, we can kinda control the number of pasted signs that will be rotated. By changing it, in fact, we can change the probability of the rotation degrees in the range to be selected, changing basically the probability that a sign will be rotated or not.

Realistic

No realistic version of rotation has been applied because the signs can appear very rotated in any possible position of the image. Rotation is not something that is really affected by the context in which the sign is pasted, e.g. a sign can be rotated with respect to the camera in any position in which the sign appears, according to the car movement.

3.1.6 Brightness

Non-realistic

The brightness level of the pasted signs can be controlled in the configuration file by defining the minimum and maximum possible brightness value of the sign, from a minimum of 0, meaning a black image, to a maximum of 1, meaning a white image. Moreover, as with rotation, we defined four possible probability distributions for the brightness levels, that are uniform, low, medium and high. The principle is similar to the rotation, but the difference is that for low, medium and high the sigma is always a very low value, such that the Gaussian distribution is very pronounced, but what changes is the mean: low probability has the mean on the minimum possible brightness value (the lower bound of the interval), medium has the mean on the average of minimum and maximum possible value (the center of the interval) and high has the mean on the maximum possible brightness value (the upper bound of the interval). This controls the probability that the signs will be darker or brighter. These three parameters (min, max and probability distribution) can be configured in the configuration file. The brightness level of the pasted sign is chosen randomly according to these parameters.

Realistic

The realistic version of this augmentation consists in computing the average brightness of the area that will be occupied by the sign. Once computed, this

brightness level is applied to the template. When doing this operation, another parameter can be configured, which is a parameter controlling the number of pixels around the area in which the sign will be pasted to also include in the computation of the average brightness level. For example, if this parameter is set to 10, a border of 10 pixels around the area in which will be pasted is also included in the estimation of the brightness value. Namely, if the patch (the area of the image in which the sign will be pasted) to consider for the computation of the brightness level goes from point (50, 50) to point (100, 100), with this parameter set to 10, the actual patch of the image used to compute the brightness level will be the patch going from point (40, 40) to point (110, 110). We also added another parameter that is a boolean allowing to hard cap the estimated level to a minimum and a maximum. If this parameter is true, the estimated brightness value will be capped to the min level defined in the brightness configuration if it is smaller than the min value, while it will be capped to the max level defined in the brightness configuration if it is bigger than the max value. The reason why we defined such parameter is because it happens that when the sign is pasted into an area of very low/very high brightness, the estimation parameter returned by the estimation is very low/very high, sometimes making the sign very dark or very bright to the point that is not possible to see colours or the icons/text in the sign. Adding this hard capping parameter prevents this situation while still estimating the brightness level based on the position in which it is pasted. In Figure 3.2 it is possible to see the effect of hard capping in realistic brightness. In Figure 3.3 it is instead possible to see the effect of different estimation border sizes (without capping) in realistic brightness.

3.1.7 Contrast

Non-realistic

The contrast level of the pasted signs can be controlled in the configuration file by defining the minimum and maximum possible contrast value of the sign, from a minimum of 0, meaning a image with a flat gray colour (the minimum contrast possible) to a maximum of 1, meaning an image with maximum possible contrast. As with brightness, we also defined four possible probability distributions for the contrast levels that are the same and work the same as for brightness. This controls the probability that the signs will have higher or lower contrast. These three parameters (min, max and probability distribution) can be configured in the configuration file. The contrast level of the pasted sign is chosen randomly according to these parameters.

Realistic

The realistic version of this augmentation consists in computing the average contrast of the area that will be occupied by the sign. Once computed, this contrast level is applied to the template. The same parameter that can control the border around the patch for the estimation to include in the estimation that we defined for the brightness was also defined for the contrast. The same capping parameter defined for brightness was also defined for contrast because the same thing that happened with brightness happened when pasting the sign in a low contrast area, so the cap parameter helps in mitigating this effect. In Figure 3.2 it is possible to see the effect of hard capping in realistic contrast. In Figure 3.3 it is instead possible to see the effect of different estimation border sizes (without capping) in realistic contrast.

3.1.8 Gaussian noise

Non-realistic

It is also possible to add Gaussian noise to the pasted signs. In the configuration file, in fact, it is possible to control the maximum and minimum possible mean and variance of the Gaussian noise to apply to the pasted signs. Then, these two parameters are chosen randomly accordingly to these min and max limits defined in the configuration. We also defined for possible probability distributions that works the same way of the probability distributions of brightness and contrast, though in this case these distributions only work on the variance range of the noise, not on the mean. The mean range has always a uniform distribution. These parameters control the amount of Gaussian noise that the traffic signs will have and can be configured in the configuration file. The amount of Gaussian noise of the pasted sign is chosen randomly according to these parameters.

Realistic

The realistic version of this augmentation consists in estimating the sigma of the noise in the area that will be occupied by the sign. The estimation is done by convolving this area with a particular 3x3 kernel. Once computed, this sigma level is used as sigma value of the Gaussian noise that is applied to the template. The parameter that controls the border around the pasting area to include the estimation that we defined for brightness and contrast has also been defined for the Gaussian noise.

3.1.9 Motion blur

Non-realistic

It is also possible to add some motion blur to the pasted signs. Since images of the domain of traffic sign are usually taken by moving cars, many signs may

appear blurred because of the motion. Moreover, considering that in our case the images were taken by phones mounted on moving cars, the only kind of blur that these cameras can capture is motion blur (due to the motion of the car) since the focus of phone cameras is usually to infinity. In the configuration file, it is possible to control the minimum and maximum possible amount of motion blur that will be applied to the traffic signs. These two parameters control the size of the kernel that will be applied when applying motion blur to the sign. Moreover, it is possible to control the probability distributions of the various blur levels in the defined range, as with the previous augmentations. The amount of motion blur is chosen randomly according to these parameters and the direction of the motion blur is chosen randomly between four possible directions, horizontal, vertical and the two diagonals.

Realistic

In case of blur, estimation algorithms are more complicated that the previous ones. It is possible to use variance of Laplacian method to detect whether an image is blurry or not and its blur level, however this method may be affected by noisy images. Moreover, this method does not return as a blur value the size of the blur kernel, since the more blurry the image, the closer to 0 the returned value will be, so it is nearly impossible to use this method to estimate a kernel size to apply to the template as we did for previous methods. More complex estimation algorithms involve the use of particular transforms (e.g. the Fourier transform) or the use of Deep Neural networks just to estimate the size of the blur kernel, however these are very complex algorithms and we wanted to keep things as simple as possible for the realistic implementations. Moreover, motion blur requires the estimation not only of the kernel size but also of the blur direction, which is something not easy to do. Since we wanted to keep things simple for the realistic augmentations, what we did was not implementing any estimation algorithm, but rather applying some motion blur to the traffic signs based on their dimension in the image. The smaller the

template, the smaller the kernel size of the blur filter will be so the less it will be blurred. This is based on the assumption that when capturing images from a moving car, the images will be in most cases blurry, but in some cases the blur will be so small that is nearly not visible to a human eye. Phone cameras have focus to infinity so the blur of the images mainly concerns motion and, in particular, the amount of blur a sign has is mainly related to the distance of the sign with respect to the camera: signs that are closer to the camera, so that appear bigger in the image, will be more blurry than signs appearing very distant from the camera, namely signs that appear smaller in the images, because they will move faster with respect to the camera compared to signs that are far from the camera. What we did was thus compute the kernel size based on the dimension of the sign in the image such that smaller signs will have motion blur with a smaller kernel sizes and will appear less blurred than bigger signs. Moreover, depending on the position of the sign in the image with respect to the point at infinity corresponding to the road, the direction of the blur will change. In general, if we consider a car moving on a road, the car will move towards the point at infinity given by the road lines. For this reason, no matter the position of the sign in the image, the motion blur direction of the sign will point to this point at infinity. In order to decide the direction of the motion blur, thus, we considered the center of the image as the point at infinity corresponding to the road and based on the position of the template with respect to this point, we decided the direction of the motion blur to apply. The reason why we considered the center of the image as this point to infinity is that we can imagine that in most cases images will be captures from a car moving on a straight road and since the camera is placed usually on the dashboard of the car pointing towards the road, we can assume that the points at infinity of the road will be placed more or less around the center of the image. In case the sign is placed at the center of the image, no blur is applied, since the car is supposed to be moving towards the sign and thus the camera will not capture motion blur on that sign. This realistic motion blur is also

based on the assumption that in general even if not visible immediately to the human eye, blur is still present in the images, so this way the blur will not pop out in the image but will still be there, imitating what happens in reality. The range of the possible motion blur kernel size was defined by using the same min and max parameters for the kernel size in the augmentation configuration that are used for the non-realistic augmentation. The amount of motion blur applied to the signs will thus be related to its size in the image from a min to a max defined into the configuration file. These two parameters are not an hard cap on the motion blur but rather the definition of the range of possible kernel sizes to select based on the dimension of the various signs. We did not add the estimation border and hard cap parameters for this realistic augmentations since there is no estimation, the only information from the image that is used is the position in which the sign will be pasted.

3.1.10 Edge blur

In order to better blend the image with the background, we blurred the alpha channel (the transparency channel of the template) with a Gaussian blur with kernel size of 3x3. There is no realistic or unrealistic version to this augmentation since it is done just to blend the template with the background.

3.2 Domain transfer

When using domain-related background images, we may end up in having traffic signs that are already present and annotated in the image. Depending on the domain of these traffic signs and on the target domain of our traffic signs, we would like to remove or replace these signs in order to have a dataset composed only of signs in which we are interested. For example, if we are using background images of streets and roads from Italy but we would like to generate a dataset with Swedish traffic signs, we must replace all the Italian signs in the background images with the corresponding Swedish signs before actually

Without hard cap

With hard cap

Figure 3.2: Qualitative results of our realistic copy-paste augmentation without hard capping (top) and with hard capping (bottom).

augmenting it. In fact, while Italian signs have a white background, Swedish signs have a yellow background and present some differences in terms of used font, images and proportions of the various elements characterizing the sign icon. For this reason, when moving from one country to another, we must replace the signs in order to perform a domain transfer from the domain of one country to our desired target domain. We thus implemented a domain transfer technique that allows to specify the source domain and the target domain (source and target countries) that uses the annotated signs in the background

images and finds the equivalent signs for the target domain. Then, we apply augmentations to these signs such that they imitate the appearance of the original signs in order to replace it as accurately as possible.

The configuration file thus includes parameters that we can use to control the domain transfer. First, we have the source and target domains: by specifying these two parameters, the algorithm uses the annotation of the original signs and a mapping to find the equivalent signs for the target domain. We also need to transfer the photometric and geometric features of the original sign to the target sign. For the size of the sign, we scale the target sign template accordingly to the original one, by using the annotation size as a measure for it, in order to meet the original sign dimension, and we use the annotation coordinates as the position for the pasting. The homography, namely the orientation of the sign with respect to the camera, was not transferred. To transfer the various photometric features such as brightness, contrast, noise and blur, we use the realistic version of the augmentations that we described in the previous section and we use as the area of the estimation the annotation area without any estimation border such that we perfectly meet the features of the original sign. As said before for realistic blur, we did not use any estimation algorithm, thus the blur for the transferred signs was computed as described before, based on the signs position with respect to the point at infinity of the road and with respect to its size in the image (namely, the distance from the camera).

This must be done also when transferring from countries whose signs are similar (e.g. from Switzerland to Italy) since the signs might present some differences even if they look very close. Moreover, when moving from one country to another, we may end up in signs that are shared between the two countries and for this reason they are very similar if not the same sign (for example, the parking sign is basically the same in all countries). To address this situation, we decided to include another parameter in the configuration that allows to choose if the shared signs must be replaced anyway by a target domain template or if the original version can be kept.

In case we do not want to apply domain transfer for various reasons, it is possible to configure a hydra configuration file for the training that allows to exclude some sign classes from the loss, such that we can train on a non-transferred dataset by ignoring certain classes and considering only others. For example, if we are augmenting an italian dataset with Swedish traffic signs but we do not apply domain transfer, we will have both white and yellow signs in the image, but since our target domain is Sweden, we want to train a detector only for yellow signs. What we do in this case is thus configuring this file by specifying only the yellow sign classes as the classes to use, while ignoring the white classes. The detector will still find white signs, because they will remain annotated and their annotation cannot be removed since it may create noise, but once it finds one of these ignored signs in the training samples, it will ignore them in the loss.

We added the hard cap parameter we described when talking about realistic augmentations also to the domain transfer phase: it is highly unexpected that when transferring from one country to another we will end up in signs being nearly unreadable, because we are using exactly the patches of the images corresponding to the original signs and these signs are usually readable, otherwise they will not be annotated at all, but since we cannot assume that all the annotations are perfectly readable, we decided to add the possibility of a hard capping also when transferring signs from one country to another. There is no need to add the estimation border parameter however because we want to transfer exactly the features of the patch corresponding to the sign.

We also applied the same edge blurring technique described before in order to better blend the transferred signs with the background.

The transferring operation also changes the existing annotations in other to meet the dimensions of the new sign and the new class.

Estimation border: 0px

Estimation border: 50px

Estimation border: 100px

Figure 3.3: Qualitative results of our realistic copy-paste augmentation with different estimation borders.

Without domain transfer

With domain transfer

Figure 3.4: Results of our domain transferring operation. On top, we can see the original image with a white 60 speed limit sign from Switzerland, while on bottom we can see the domain-transferred image with a yellow 60 speed limit sign from Sweden. We can also see how the annotation was changed accordingly to the new sign.

Chapter 4

Material

For the copy-paste augmentation, we need background images and sign templates in order to compose the synthetic data that we will use for the training operation. Moreover, we need a network to test if this technique is actually efficient and a test set consisting of real data that we can use in combination with this network to test our model performances when trained on synthetic data.

4.1 Datasets

The organic datasets that we used consist in road and street images with annotated traffic signs in it. We used custom built datasets consisting of images taken in a single country by phones mounted on moving vehicles. We used three datasets in total:

- empty traffic sign dataset: a dataset split into train, validation and test sets that contains non-annotated images of roads and streets without any traffic sign
- Sweden traffic sign dataset: a dataset split into train, validation and test sets that contains annotated images of road and streets with Swedish traffic signs

- Switzerland traffic sign dataset: a dataset split into train, validation and test sets that contains annotated images of road and streets with Swiss traffic signs

4.1.1 Sign classes

The most notable difference between the signs of different countries is the background color, which is either white or yellow. For example, Sweden uses signs with yellow background while Switzerland uses white background. However, some signs are shared between all the countries, like the parking sign for example, so we have a single class for all the countries for these shared signs. These classes are organized in a parent-child structure in order to group classes by category, where the parent categories are the following

- speed_sign
- warning_sign
- roadwork_sign
- prohibitory_sign
- information_sign
- traffic_camera_sign
- pedestrian_crossing

The sign annotation consists in an id which represents the class of the sign and a bounding box representing the area of the image that contains the sign.

4.1.2 Training data

The training data can be, depending on the experiment, either synthetic data or real data. When using real data, we simply used one of the two annotated sets mentioned before, while when using synthetic data, we augmented one of

the three sets by copy-pasting sign templates, the actual sign icons, on their images. In synthetic data generation, the images in these datasets are used as a background image for the copy-paste operation, preserving the annotations that are already there, if any. More details in Chapter 5.

Background images

Background images are thus JPEG images of road and streets that may or may not contain already annotated signs.

Template images

As templates, we decided to use sign icons taken from the web, mainly from Wikipedia. Wikipedia provides pages containing vector icon of signs for each country, so we can download all the icons from Wikipedia and use them as templates for the augmentation. For example, we can find all the traffic signs of Sweden on Wikipedia [61]. However, these Wikipedia pages do not provide all the traffic signs from the various countries, for example they only provide usually one example for the speed limit signs, but we need all of them. Fortunately, we have equivalent pages from Wikimedia that instead contain all the signs the remaining signs [62]. To find and define equivalences between signs for the domain transfer, we used again Wikipedia which has a page showing all the equivalences between various European countries [63].

The template images that we retrieved satisfy the requirement of transparent border and high resolution, which are mandatory requirements for a high-quality copy-pasting. The templates were then grouped and renamed accordingly to the classes defined in Section 4.1.1.

If templates of some signs were not available, we recreated them through image editing programs.

4.1.3 Validation and testing data

The validation and testing data consists, as the training data, in images of roads and streets with annotated signs. We also used, in some experiments, synthetic validation datasets in which the signs in the dataset were all synthetic copy-pasted signs. More details are available in Chapter 5.

4.2 Model

The used model for traffic sign recognition is an SSD-like detector without any FPN using MobileNetV3 as backbone. that was developed by Univrses. We used this network to test the augmentation technique and its effectiveness in the task of traffic sign recognition. The reason why MobileNetV3 was chosen is because the images used to train our detectors were taken by smartphones mounted on moving cars, meaning that in order to deploy this system on mobile phones, we need an efficient and lightweight network, so it makes sense to test copy-paste augmentation on this network. Moreover, MobileNetV3 was designed for mobile deployment and, although being small, it proved to be able to perform the chosen task very well when provided with enough good quality data.

4.3 Tools

The programming language we used is Python, in combination with TensorFlow library to create the model and manage datasets. For the copy-paste augmentation, we used libraries as Numpy, Scipy, PIL, OpenCV, Pandas and Hydra to code the augmentation methods. For the training, the tuning of the various parameters and the model testing, we used Wandb. As IDE, we used PyCharm, in combination with the collaborative developing tools Git and GitHub.

Chapter 5

Experiments and results

5.1 Training with real yellow signs

As the baseline, namely the model use for all the comparisons, we trained a model detecting yellow Swedish signs that was trained only on real yellow Swedish traffic signs. We did this because we wanted to compare models trained with synthetic data with models trained on real data. The baseline experiment thus consisted in training our model using the Swedish non-augmented training set, while evaluating it on the Swedish non-augmented validation set. The training set contains 8360 images while the validation set contains 1874 images. We trained our model for 100 epochs with early stopping and with a batch size of 16. While training the model, we also tuned some training hyperparameters as the initial learning rate, the decay steps and the regularization l2 factor. We launched a grid hyperparameter search over these hyperparameters, while using 0.01, 0.001 and 0.0001 as initial learning rate values. For the learning rate decay, we used exponential decay with step values 522, 1045, 2090, 4180 and 8360 which are, respectively, the number of steps for each training epoch (the number of batches) multiplied by 1, 2, 4, 8 and 16. The first number was actually 522,5 (8360/16 = 522,5) so we rounded it. For the regularization l2 factor, we used values 1e-4, 1e-6 and 1e-8.

We chose the best hyperparameters based on the best validation f1 score

averaged per class that the model was reaching on the Swedish validation set. We kept the training seeds to 0 among all these runs to enable reproducibility and avoid measuring differences in the results due to different seeds. In Table 5.1 it is possible to see the first 10 runs with the highest best val f1 score. As it is possible to see, the model with the highest best val f1 score was the one having initial learning rate of 0.001, decay steps of 4180 and regularization l2 factor of 1e-8. However, when looking at their val f1 score graph we could see that all these models were more or less noisy. Models trained with a higher learning rate were more noisy, especially in the first epochs so, since the difference in best val f1 score among these models is very small, we decided to choose the hyperparameters corresponding to the model which was the less noisy among all of these. In the table, it is possible to see the chosen model as the one in bold. We did this because we will be using these hyperparameters also for the tests with synthetic data and since these tests are usually more noisy than the ones with real data, especially in the first epochs, we decided then to pick the hyperparameters producing the less noise. In Figure 5.1 we can see the difference in noise between the best model, the first in the table, and the model that we selected. The orange model is the one that we selected while the brownish one is the best one. The chosen hyperparameters where thus an initial learning rate of 0.001, a number of decay steps of 2090 and a regularization l2 factor of 1e-8.

As it is possible to see from Table 5.1, the hyperparameter that was influencing the most the performances was the initial learning rate, as expected.

After doing this and selecting the best model, we launched the training again for 4 times by keeping everything the same while only changing the training seeds in order to measure better the performances of the model and get a better sense of the noise between different runs. We executed 4 runs with seeds 0, 1507, 2307 and 3425. The results of these experiments are in Table 5.2.

best val f1	initial learning rate	decay steps	regularization l2 factor
0.7697	0.001	4180	1e-8
0.7646	0.001	2090	1e-4
0.7638	0.001	8360	1e-8
0.7636	0.001	4180	1e-4
0.7626	0.01	522	1e-8
0.7621	**0.001**	**2090**	**1e-8**
0.7567	0.001	1045	1e-6
0.7552	0.001	2090	1e-6
0.7545	0.001	4180	1e-6
0.7534	0.01	4180	1e-4

Table 5.1: Results of the sweep on the model trained on real yellow Swedish signs. These are the first 10 most performing models in terms of best val f1 score averaged per class.

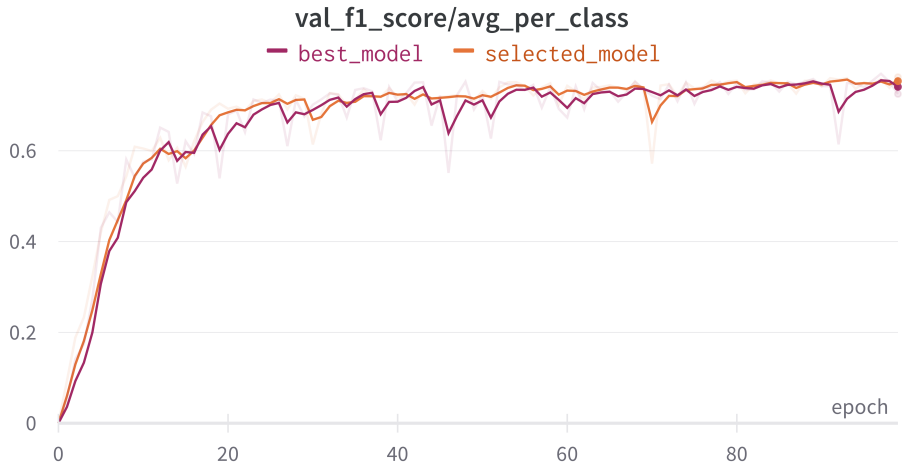


Figure 5.1: Val f1 score averaged per class of the best model compared to the selected one. An exponential moving average smoothing of 0.3 is applied to the graph to make it more clear.

training seed	best val f1
0	0.7545
1507	0.7488
2307	0.7560
3425	0.7709
mean	**0.7575**
variance	**0.0088**

Table 5.2: Results of the seeds sweep on the model trained on real yellow Swedish signs. The best val f1 score is averaged per class.

5.2 Training with synthetic yellow signs

We also created synthetic datasets and trained models with synthetic data. In these experiments, we created the training sets containing only copy-pasted yellow Swedish traffic signs, without any real sign. We used as base for our synthetic training sets a dataset consisting of images of Swedish roads and streets which did not contain any traffic sign. We also made sure that all the images did not contain any traffic sign that was not annotated in order to avoid introducing noise while training and in order to avoid using any real Swedish yellow traffic sign. This dataset of empty images consists in 13030 images, but in order to keep consistency and to have a fair comparison with the baseline, we augmented and kept only the first 8360 images in order to have the same number of training images as with the baseline. As with the baseline experiment, we trained our models for 100 epochs with early stopping and with a batch size of 16, while keeping the same hyperparameters that we selected from the baseline experiment in order to compare these models as fairly as possible. We thus used initial learning rate of 0.001, 2090 decay steps and 1e-8 as regularization l2 factor.

While training, we evaluated the model using the same validation set used for the baseline experiment.

To avoid measuring only noise, we also executed multiple runs for each experiment by only changing the training seeds between the various runs. We launched 4 runs per experiment with seeds 0, 2307, 1507, 3425, as we did with the baseline. As the seed for the augmentation phase, instead, we used 1507 for all experiments.

5.2.1 Realistic copy-pasted training data

As first experiment, we created a synthetic training set by applying copy-paste augmentation with all sign augmentations. While applying per-sign augmentations, we used realism where possible, so we used the following configuration:

- signs/background ratio to 1.6 % of the whole image
- no overlap padding to 5px
- scale of the pasted signs from 5 % to 12 % of the original template icon size (all the icons are standardized in size), using step size 1 %
- rotation of the pasted signs over axis x from -30° to 30°, with step size 1° and probability distribution to low
- rotation of the pasted signs over axis y from -60° to 60°, with step size 1° and probability distribution to low
- rotation of the pasted signs over axis z from -10° to 10°, with step size 1° and probability distribution to low
- **realistic contrast** of the pasted signs computed using an estimation neighborhood of 50px, minimum capping to 10 % and maximum capping to 90 %
- **realistic brightness** of the pasted signs computed using an estimation neighborhood of 50px, minimum capping to 10 % and maximum capping to 90 %
- **realistic motion blur** of the pasted signs
- **realistic gaussian noise** of the pasted signs computed without the use of an estimation neighborhood (set to 0px)
- edge blurring enabled

The augmentations to the single traffic sign were applied in the following order: scale, rotation, contrast, brightness, motion blurring, gaussian noise and edge blurring. Since we were not performing domain transfer, the used estimation patch for the realistic augmentations was the patch of the image in which the sign was pasted. We also did not use a custom probability distribution for the sign classes so each sign class had the same probability of being pasted to the image.

The results of this experiment are in Table 5.3.

training seed	**best val f1**
0	0.3638
1507	0.4197
2307	0.4083
3425	0.4117
mean	**0.4009**
variance	**0.0634**

Table 5.3: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with realism. The best val f1 score is averaged per class.

5.2.2 Ablation studies on realistic augmentations

We executed some ablation studies to see how much realism actually contributes to the performances of the whole system. We run these tests by disabling realism one by one for the augmentations that have the possibility of being also realistic. We created these training sets by following the same procedure of Section 5.2 but we disabled realistic augmentations one by one by turning one of the realistic augmentations to its non-realistic version in each experiment. More specifically, we did the following experiments:

- a training with all the realistic augmentations but with **non-realistic brightness**, using a minimum value of 10 %, a maximum value of 90 %, a step size of 5 % and probability distribution to medium

- a training with all the realistic augmentations but with **non-realistic contrast**, using a minimum value of 10 %, a maximum value of 90 %, a step size of 5 % and probability distribution to medium
- a training with all the realistic augmentations but with **non-realistic motion blur**, using a minimum kernel size of 4, a maximum kernel size of 10, a step size 1 and probability distribution low
- a training with all the realistic augmentations but with **non-realistic gaussian noise**, using minimum and maximum mean of 0, minimum variance of 5, maximum variance of 60, step size for variance of 1 and probability distribution high

The results of these experiments are, respectively, in Tables 5.4, 5.5, 5.6 and 5.7.

Another experiment that we did was training a model using a dataset with only synthetic copy-pasted signs without any form of realism. In this setting, we were creating the synthetic training set following the same procedure of Section 5.2 but we disabled all realistic augmentations by setting brightness, contrast, motion blur and gaussian noise as follows:

- **non-realistic brightness** with minimum value of 10 %, maximum value of 90 %, step size of 5 % and probability distribution to medium
- **non-realistic contrast** with minimum value of 10 %, maximum value of 90 %, step size of 5 % and probability distribution to medium
- **non-realistic motion blur** with minimum kernel size of 4, a maximum kernel size of 10, step size 1 and probability distribution low
- **non-realistic gaussian noise** with minimum and maximum mean of 0, minimum variance of 5, maximum variance of 60, step size for variance of 1 and probability distribution high

The results of this experiment are in Table 5.8.

training seed	best val f1
0	0.3585
1507	0.4014
2307	0.3733
3425	0.3938
mean	**0.3818**
variance	**0.0521**

Table 5.4: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with realism but with non-realistic brightness. The best val f1 score is averaged per class.

training seed	best val f1
0	0.4385
1507	0.4178
2307	0.4506
3425	0.4046
mean	**0.4279**
variance	**0.0424**

Table 5.5: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with realism but with non-realistic contrast. The best val f1 score is averaged per class.

training seed	best val f1
0	0.3884
1507	0.4031
2307	0.4098
3425	0.4027
mean	**0.4010**
variance	**0.0081**

Table 5.6: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with realism but with non-realistic motion blur. The best val f1 score is averaged per class.

5.2.3 Ablation studies on non-realistic augmentations

Just as we did with realism ablation studies, we executed some ablation studies on each sign augmentation, this time by starting from a setting with all augmentations in their non-realistic version and proceeding by completely disabling one by one all augmentations, including the ones that never had a realistic version. In this setting, we were creating the synthetic training set

training seed	best val f1
0	0.4054
1507	0.4024
2307	0.4085
3425	0.3970
mean	**0.4033**
variance	**0.0024**

Table 5.7: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with realism but with non-realistic gaussian noise. The best val f1 score is averaged per class.

training seed	best val f1
0	0.3929
1507	0.4181
2307	0.4302
3425	0.3951
mean	**0.4091**
variance	**0.0328**

Table 5.8: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism. The best val f1 score is averaged per class.

following the same procedure of the final experiment of Section 5.2.1, using only non-realistic augmentation, but we completely disabled them one by one for each experiment. More specifically, we did the following experiments:

- a training with all non-realistic augmentations but **without scale** augmentation
- a training with all non-realistic augmentations but **without rotation** augmentation
- a training with all non-realistic augmentations but **without contrast** augmentation
- a training with all non-realistic augmentations but **without brightness** augmentation

- a training with all non-realistic augmentations but **without motion blur** augmentation
- a training with all non-realistic augmentations but **without gaussian noise** augmentation
- a training with all non-realistic augmentations but **without edge blur-ring** augmentation

The results of these experiments are, respectively, in Tables 5.9, 5.10, 5.11, 5.12, 5.13, 5.14 and 5.15.

training seed	**best val f1**
0	0.2802
1507	0.2904
2307	0.2969
3425	0.2846
mean	**0.2880**
variance	**0.0052**

Table 5.9: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without scale. The best val f1 score is averaged per class.

training seed	**best val f1**
0	0.3755
1507	0.3518
2307	0.3570
3425	0.3392
mean	**0.3559**
variance	**0.0227**

Table 5.10: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without rotation. The best val f1 score is averaged per class.

training seed	best val f1
0	0.3962
1507	0.4267
2307	0.3991
3425	0.3947
mean	**0.4042**
variance	**0.0276**

Table 5.11: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without contrast. The best val f1 score is averaged per class.

training seed	best val f1
0	0.1613
1507	0.1409
2307	0.1612
3425	0.1896
mean	**0.1633**
variance	**0.0401**

Table 5.12: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without brightness. The best val f1 score is averaged per class.

training seed	best val f1
0	0.3923
1507	0.3912
2307	0.4040
3425	0.3870
mean	**0.3936**
variance	**0.0053**

Table 5.13: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without motion blur. The best val f1 score is averaged per class.

5.3 Training with real and synthetic yellow signs

Another experiment that we designed was to train models with a mixture of real signs and synthetic signs. In this setting, we augmented a dataset containing already annotated traffic signs. In particular, we augmented the same

training seed	best val f1
0	0.4104
1507	0.4140
2307	0.4198
3425	0.3942
mean	**0.4096**
variance	**0.0120**

Table 5.14: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without gaussian noise. The best val f1 score is averaged per class.

training seed	best val f1
0	0.4072
1507	0.4203
2307	0.4175
3425	0.4095
mean	**0.4136**
variance	**0.0039**

Table 5.15: Results of the seeds sweep on the model trained on copy-pasted yellow Swedish signs with no realism and without edge blurring. The best val f1 score is averaged per class.

dataset that we used to train the baseline, namely the training set of real non-augmented Swedish yellow signs. This dataset consists of 8360 training images. We trained our models for 100 epochs with early stopping and with a batch size of 16, while keeping the same hyperparameters that we selected from the baseline experiment in order to compare these models as fairly as possible, so we used initial learning rate of 0.001, 2090 decay steps and 1e-8 as regularization l2 factor.

While training, we evaluated the model using the same validation set used for the baseline, namely the validation set of real non-augmented Swedish yellow signs, consisting in 1874 validation images.

In order to avoid measuring noise, we also executed multiple runs for each experiment by only changing the training seeds between the various runs. We launched 4 runs per experiment with seeds 0, 2307, 1507, 3425. As augmentation seed, instead, we used 1507 for all experiments.

We created the training set by using the same configuration that we used for the experiment described in Section **??**, so we used all the augmentations in their non-realistic way.

The results of this experiments are in Table 5.16.

training seed	best val f1
0	0.7438
1507	0.6965
2307	0.7123
3425	0.7433
mean	**0.7240**
variance	**0.0552**

Table 5.16: Results of the seeds sweep on the model trained on real and copy-pasted yellow Swedish signs with no realism. The best val f1 score is averaged per class.

5.4 Training with domain-transferred yellow signs

The final training experiment consisted in training a model using a dataset of domain transferred Swedish yellow signs. In this setting, we created the training set by augmenting a dataset of white signs. We applied domain transfer to these images in order to convert the white signs to yellow signs. The dataset of white signs that we used consisted in 2624 images with annotated white sign from Switzerland. However, some of these images did not contain any traffic sign. While augmenting this dataset, we skipped the empty images and we kept only the images containing annotated traffic signs. By skipping the empty images, we obtained a dataset of 1747 images with real white signs. In order to do a fair comparison with the baseline, we needed however 8360 training images. We only have 1747 images in this dataset, so in order to have a higher number of images we simply augmented each image in the training set 5 times so we obtained 1747*5=8735 augmented training images, of which we used the first 8360 images.

While domain transferring, we both replaced white signs, including signs

of shared classes between yellow and white domains, and copy-pasted new synthetic yellow signs. In this setting, we were using copy-pasting to both replace the white signs in the images and, at the same time, pasting new signs. Since we were replacing all the white signs we were thus using only synthetic data to train the model.

We trained our models for 100 epochs with early stopping and with a batch size of 16, while keeping the same hyperparameters that we selected from the baseline experiment in order to compare these models as fairly as possible, so we used initial learning rate of 0.001, 2090 decay steps and 1e-8 as regularization l2 factor.

We evaluated the model using the same validation set used for the baseline, namely the validation set of real non-augmented Swedish yellow signs, consisting in 1874 validation images.

In order to avoid measuring noise, we also executed multiple runs for each experiment by only changing the training seeds between the various runs. We launched 4 runs per experiment with seeds 0, 2307, 1507, 3425. As augmentation seed, instead, we used 1507 for all experiments.

We created the training set by using the same configuration that we used for the experiment described in Section **??**, so we used all the augmentations in their non-realistic way. When applying domain transfer, we replaced signs also for common classes and we did not use any capping on the augmentation parameters for the replaced signs.

The results of this experiments are in Table 5.17.

5.4.1 Ablation studies on domain transfer

The second experiment was about running ablation studies on domain transfer, namely by training models by using only replacement when creating the domain-transferred training set and then by using only copy-pasting. When using only replacement, we are training with fully-synthetic data since all the

training seed	best val f1
0	0.4860
1507	0.4838
2307	0.4790
3425	0.4563
mean	**0.4763**
variance	**0.0185**

Table 5.17: Results of the seeds sweep on the model trained on domain-transferred data using replacement of white signs and pasting of new yellow signs. The best val f1 score is averaged per class.

real signs have been replaced, while when training with only copy-pasting we were using a mix of real and synthetic data to train our model because we were not replacing the original white signs. We were however ignoring all the white classes in the loss, in order to avoid introducing noise, while keeping the classes that were shared between yellow and white domains.

The results of these experiments are available, respectively, in Tables 5.18 and 5.19.

training seed	best val f1
0	0.3071
1507	0.3115
2307	0.3044
3425	0.3132
mean	**0.3090**
variance	**0.0016**

Table 5.18: Results of the seeds sweep on the model trained on domain-transferred data using only replacement of white signs. The best val f1 score is averaged per class.

Table

5.5 Validating with synthetic yellow signs

Another experiment that we designed was about using copy-pasted data for validating a model trained on some training data. We tested both models

training seed	best val f1
0	0.4359
1507	0.4459
2307	0.4442
3425	0.4381
mean	**0.4410**
variance	**0.0023**

Table 5.19: Results of the seeds sweep on the model trained on domain-transferred data using only copy-pasting of new yellow signs. The best val f1 score is averaged per class.

trained on real data and on synthetic data. In this setting, we created a validation set by augmenting the dataset of empty Swedish road images. In order to have comparable results with the validation set of real yellow Swedish images, we augmented only 1874 images. We also used the same number of training images that we used in the baseline both with real and synthetic training data, namely we used 8360 training images. Moreover, we also created a validation set by domain transferring from white signs to yellow Swedish signs.

We trained our models for 100 epochs with early stopping and with a batch size of 16, while keeping the same hyperparameters that we selected from the baseline experiment in order to compare these models as fairly as possible, so we used initial learning rate of 0.001, 2090 decay steps and 1e-8 as regularization l2 factor.

In order to avoid measuring noise, we also executed multiple runs for each experiment by only changing the training seeds between the various runs. We launched 4 runs per experiment with seeds 0, 2307, 1507, 3425. As augmentation seed, instead, we used 1507 for all experiments when creating the training set, while we used 4468 as seed when augmenting to create validation data.

5.5.1 Realistic copy-pasted validation data with non-realistic copy-pasted train data

In this experiment, we trained a model with non-realistic copy-pasted yellow signs and we validated the same model on realistic copy-pasted yellow signs. We created the training data by following the configuration used in the experiment described in Section **??**, while we created validation data by following the configuration used in the experiment described in Section 5.2. We created both training and validation sets by augmenting the set of empty Swedish road images. In order to avoid using the same background images both for training and validation, we augmented the first 8360 training images to create the training set while we augmented the following 1874 training images to create the validation set.

The results of this experiment are in Table 5.20.

training seed	best val f1
0	0.9775
1507	0.9760
2307	0.9768
3425	0.9775
mean	**0.9770**
variance	**0.0001**

Table 5.20: Results of the seeds sweep on the model trained on non-realistic copy-pasted yellow signs and validated on realistic copy-pasted yellow signs. The best val f1 score is averaged per class.

5.5.2 Non-realistic copy-pasted validation data with non-realistic copy-pasted train data

In this experiment, we trained a model with non-realistic copy-pasted yellow signs and we validated the same model on non-realistic copy-pasted yellow signs. We created the training data by following the configuration used in the final experiment described in Section 5.2.2, while we also created validation data by following the configuration used in the last experiment described in

Section 5.2.2. We created both training and validation sets by augmenting the set of empty Swedish road images. In order to avoid using the same background images both for training and validation, we augmented the first 8360 training images to create the training set while we augmented the following 1874 training images to create the validation set.

The results of this experiment are in Table 5.21.

training seed	**best val f1**
0	0.9923
1507	0.9929
2307	0.9920
3425	0.9941
mean	**0.9928**
variance	**0.0001**

Table 5.21: Results of the seeds sweep on the model trained on non-realistic copy-pasted yellow signs and validated on non-realistic copy-pasted yellow signs. The best val f1 score is averaged per class.

5.5.3 Domain-transferred validation data with non-realistic copy-pasted train data

In this experiment, we trained a model with non-realistic copy-pasted yellow signs and we validated the same model on domain-transferred data from the domain of white signs. We created validation data by only replacing white signs and not by copy-pasting new yellow signs, in the same way we were doing when performing ablation studies for domain transfer in Section 5.4. Since we needed 1874 images, we augmented each image in the white traffic signs dataset 2 times and then kept only the first 1874 augmented images.

The results of this experiment are in Table 5.25.

training seed	best val f1
0	0.4634
1507	0.4686
2307	0.4701
3425	0.4654
mean	**0.4669**
variance	**0.0009**

Table 5.22: Results of the seeds sweep on the model trained on non-realistic copy-pasted yellow signs and validated on domain-transferred data from the domain of white signs. The best val f1 score is averaged per class.

5.5.4 Realistic copy-pasted validation data with real train data

In this experiment, we trained a model with real Swedish yellow signs and we validated the same model on realistic copy-pasted Swedish yellow signs. We created validation data by following the configuration used in the experiment described in Section 5.2. We created the validation set by augmenting the first 1874 training images of the set of empty Swedish road images.

The results of this experiment are in Table 5.23.

training seed	best val f1
0	0.6706
1507	0.7196
2307	0.6536
3425	0.6978
mean	**0.6854**
variance	**0.0851**

Table 5.23: Results of the seeds sweep on the model trained real yellow signs and validated on realistic copy-pasted yellow signs. The best val f1 score is averaged per class.

5.5.5 Non-realistic copy-pasted validation data with real train data

In this experiment, we trained a model with real Swedish yellow signs and we validated the same model on non-realistic copy-pasted yellow signs. We

created validation data by following the configuration used in the experiment described in Section **??**. We created the validation set by augmenting the first 1874 training images of the set of empty Swedish road images.

The results of this experiment are in Table 5.24.

training seed	best val f1
0	0.7129
1507	0.7179
2307	0.6877
3425	0.6835
mean	**0.7005**
variance	**0.0303**

Table 5.24: Results of the seeds sweep on the model trained on real yellow signs and validated on non-realistic copy-pasted yellow signs. The best val f1 score is averaged per class.

5.5.6 Domain-transferred validation data with real train data

In this experiment, we trained a model with real Swedish yellow signs and we validated the same model on domain-transferred data from the domain of white signs. We created validation data by using the same configuration used in experiment described in Section 5.4 to create the validation set and since we needed 1874 images, we augmented each image in the white traffic signs dataset 2 times and then kept only the first 1874 augmented images.

The results of this experiment are in Table 5.25.

training seed	best val f1
0	0.5332
1507	0.5199
2307	0.5137
3425	0.5068
mean	**0.5184**
variance	**0.0126**

Table 5.25: Results of the seeds sweep on the model trained on real yellow signs and validated on domain-transferred data from the domain of white signs. The best val f1 score is averaged per class.

Chapter 6

Conclusion

6.1 Discussion of experiments results

As is it possible to see, we run multiple experiments in various training/validation settings. In this section, we are going to discuss all the results we obtained for each category of experiments. Moreover, we are going to compare the various results with the baseline method and between them. Finally, we will list some possible future improvements.

6.1.1 Training experiments

These experiments have been performed on various different training sets, consisting however in the same number of images, namely 8360, while using the same validation set consisting of 1874 images with real yellow Swedish signs.

As we can see from Table 6.1, the best validation f1 score on average over the 4 experiments is achieved when training using only real data, with an average best val f1 score of 0.7575. It is possible to see that this average value drops to 0.4009 when training using only realistic copy-pasted yellow signs, while it drops to 0.4091 when training using only non-realistic copy-pasted

Experiment	Validation F1-score
baseline (real training data)	0.7575 ± 0.0094
realistic copy-paste training data	0.4009 ± 0.0252
with non-realistic brightness	0.3818 ± 0.0228
with non-realistic contrast	0.4279 ± 0.0206
with non-realistic motion blur	0.4010 ± 0.0090
with non-realistic gaussian noise	0.4033 ± 0.0049
non-realistic copy-paste training data	0.4091 ± 0.0181
without scale	0.2880 ± 0.0072
without rotation	0.3559 ± 0.0151
without contrast	0.4042 ± 0.0166
without brightness	0.1633 ± 0.0200
without motion blur	0.3936 ± 0.0073
without gaussian noise	0.4096 ± 0.0110
without edge blurring	0.4136 ± 0.0062
non-realistic copy-paste and real training data	0.7240 ± 0.0235
domain transferred training data	0.4763 ± 0.0136
by only replacement	0.3090 ± 0.0040
by only copy-pasting	0.4410 ± 0.0048

Table 6.1: Comparison in terms of best validation f1 score (averaged per class and per seed run) between the baseline and the various training experiments.

yellow signs. This is indicative of the fact that using only synthetic data created with copy-paste augmentation, both in its realistic and non-realistic form, is not entirely a valid replacement of real data, though still being a valid compromise when no data or few data is available. It is also possible to see that the difference in performance between non-realistic data and realistic data is very small, since there is no measurable difference given the standard deviation of the validation f1 score between the different runs, indicating that for the training task using a realistic or unrealistic version of the augmented data does not make too much of a difference, especially because this difference is so little that it can be due to a difference in noise between the two experiments.

It is also possible to see that when training with a combination of real data and non-realistic copy-pasted data, the average best validation f1 score does not improve, instead it drops to 0.7240. This is due to the fact that adding copy-pasted data to the real data may introduce some additional noise in the

training and add some confusion to the detector. In fact, in this scenario, copy-paste augmentation should be used only to re-balance classes that are poorly represented in the training data, by changing the probabilities that the various traffic sign classes have to be selected when copy-pasting, while in this experiment, we used the same probabilities for all the classes.

When training with domain-transferred data, we achieved an average best validation f1 score of 0.4763, which is still lower than the baseline, but is higher than the score achieved when using copy-paste augmentation without domain transfer. We suspect that this is due to the fact that while domain transferring, we are both replacing the real white signs existing in the dataset, thus imitating the distribution of signs and their photometric and geometric characteristic in reality, but at the same time we are pasting new yellow signs which are going to re-balance classes that are underrepresented in reality. These results can be further improved by pasting new signs accordingly to the distribution of the dataset used for domain transfer, in order to re-balance underrepresented classes even better and have a dataset that is even more balanced.

By looking at the results over ablation studies on realism, we can also see that realistic brightness helps compared to non-realistic brightness. In fact, we can see that, when disabling realism on brightness, the average best validation f1 score drops to 0.3818, indicating that realistic brightness helps more with respect to non-realistic brightness. We can also see how non-realistic contrast helps more than realistic contrast: in fact, we can see that when disabling realism for contrast, we obtain an average best validation f1 score of 0.4279. On the other hand, disabling realism for motion blur and gaussian noise does not change too much the performances of the detector since the average best validation f1 score goes to 0.4010 in the former case and 0.4033 in the latter case. In these two cases, the score is still higher than when using all realistic augmentations (in this case, the score is 0.4009), but the difference is so little that it might just be due to a difference in noise between the two experiments.

From ablation studies on the augmentations, we can see how when removing scale, average val f1 score drops to 0.2880, while it drops to 0.3559 when removing rotation, to 0.1633 when removing brightness, indicating that these three augmentations are the most helpful ones. On the other hand, when removing motion blur, validation f1 score drops to 0.3936. When disabling contrast, instead, we obtain an average val f1 score of 0.4042, while we obtain a score of 0.4096 when disabling gaussian noise. It is not clear, then, if motion blur, contrast and gaussian noise are useful, since the difference in the mean validation f1 score with respect to the experiment using all augmentations is more or less 0.001 , which is not significant given the standard deviation values of these three ablation studies which are 0.0073 for motion blur, 0.0166 for contrast and 0.0110 for gaussian noise. Another thing that we can see is that disabling edge blurring increases the score to 0.4136. This might be due to the fact that we are still using motion blur, so when applying motion blur and edge blurring together, the pasted signs may appear too blurry in some cases because we are blurring a border that has already been blurred by motion blur, especially when the amount of applied motion blur is already enough to blur the border and blend the sign with the background, so not applying edge blurring on top of motion blur helps while at the same time still allowing us to blend the pasted signs with the background because of the presence of motion blur. However, again, the difference is too little, so it can be due just to noise.

Regarding domain transfer, we can see how using both replacement and copy-pasting helps, because when using only replacement the average best val f1 score drops to 0.3090 while it drops to 0.4410 when using only copy-pasting. This happens because using only the original distribution of the dataset from which we are domain transferring, so applying only replacement, is not enough, especially because some traffic sign classes may be completely missing because of the domain difference. For example, some white signs may not have an equivalent yellow sign, so when transferring from white signs in this state to yellow signs, some signs may be missing and thus not replaced.

This is the reason why we apply copy-paste on top of replacement, in order to allow us also to represent also these missing classes, if any. On top of this, we are not transferring the homography of the signs when replacing, so all of our replaced signs will appear not rotated, meaning that this is another feature that will be missing if using only replacement, while it is present when using also copy-pasting. Another interesting result is the fact that when training on data that has been domain-transferred by only using copy-pasting, we achieve a score of 0.4410, which is lower than the score we achieved when we were applying both replacement and copy-pasting, but is still higher than the score we achieved when training only with realistic/non-realistic copy-pasted data. This happens because while ignoring the white sign in the loss, we are still using some real signs which are the signs of the classes shared between the white and yellow domains (for example pedestrian crossings, parking signs etc.). This limited amount of real data is used on top of copy-pasted data, so it helps in achieving a higher score with respect to using only copy-pasted data.

6.1.2 Validation experiments

These experiments have been performed on various different training and validation sets, consisting however in the same number of images, 8360 for training and 1874 for validation. We designed these experiments to see if synthetic data can be a good replacement for real validation data in cases in which the available data is too poor to create both a training and a validation set.

From Table 6.2 we can see that the use of a synthetic validation set created with copy-paste augmentation is appropriate only when training with real data. In fact, we can see that when training with real data, validating with a dataset of real yellow signs gives us an average best val f1 score of 0.7575, while when we obtain a score of 0.6854 when validating with a dataset of realistic copy-pasted yellow signs and a score of 0.7005 when validating with a dataset of non-realistic copy-pasted yellow signs. In this setting, however, a validation

Experiment	Validation F1-score
real training data	
baseline (real validation data)	0.7575 ± 0.0094
realistic copy-pasted validation data	0.6854 ± 0.0292
non-realistic copy-pasted validation data	0.7005 ± 0.0174
domain-transferred (only replacement) validation data	0.5184 ± 0.0112
non-realistic copy-pasted training data	
real validation data	0.4091 ± 0.0181
realistic copy-pasted validation data	0.9770 ± 0.0010
non-realistic copy-pasted validation data	0.9928 ± 0.0010
domain-transferred (only replacement) validation data	0.4669 ± 0.0030

Table 6.2: Comparison in terms of best validation f1 score (averaged per class and per experiment) between the various validation experiments.

set of domain-transferred yellow signs is not enough since we may not have enough samples per class in order to correctly evaluate a trained model because of the transferring operation from one domain to another. We did not test with a domain-transferred dataset with both replacement and copy-pasting because the results would have been similar to the results obtained when using only non-realistic copy-pasted validation data, since the majority of the signs in the validation dataset would have been copy-pasted signs.

On the other hand we can see that when validating with copy-pasted validation data, both realistic and non-realistic, we get a score that is close to the score that we obtained when validating with real data. There is still a noticeable gap between scores when validating with real data and synthetic data and this is due to the fact that copy-pasting does not lead to the same distribution of real data, nor it represents all the variability of real world data (e.g. occlusions).

When training with non-realistic copy-pasted data, copy-pasted data cannot be used for validation both in case of realistic and non-realistic copy-paste. In fact, as we can see, when validating with realistic copy-pasted data, we get an average best val f1 score of 0.9770, while we get a score of 0.9928 when validating with non-realistic copy-pasted data. This happens because, regardless of being realistic or not being realistic, synthetic data in both the training and validation sets will be basically the same. Moreover, when validating

with non-realistic copy-pasted data, the distribution of training data and validation data will be exactly the same, and this is the reason why we get a score that is very close to 1.00 when both training and validating with non-realistic copy-pasted data. A synthetic validation dataset is not a good replacement of a real validation set when training with synthetic data. A domain-transferred validation dataset, instead, can be a good replacement for a real validation set. In fact, we can see that when validating with a domain-transferred set that is built using only replacement, we obtain a score of 0.4669. The reason why this happens is because the data that we are using for validation is in some way closer to real data: in fact, by performing domain transfer, we are replacing real signs with some synthetic signs but we are also transferring the geometric and photometric features of the real signs to the synthetic ones, meaning that the replaced signs will be closer to real world samples than to synthetic samples. This happens only if we apply domain transfer only as replacement because if we apply it as both replacement and copy-paste, we will get similar results as when we were validating with copy-pasted data. However, this still results in an overestimation of the performances of the model on real data, since we have a difference of around 0.06 and this is due to the fact that, again, copy-pasting does not lead to the same distribution of real data neither it represents all the variability of real world data.

From the results of copy-paste augmentation on training and validation data, however, it is very hard to justify copy paste augmentation as validation set since it works so poorly for training.

6.2 Future developments

We saw how synthetic data can be used to train traffic sign detectors and, in some cases, also to evaluate them. We also identified some future developments and improvements that can be applied to this system in order to further improve the results:

- when domain transferring, it is possible to also transfer the homography (the rotation) of the source sign by computing SIFT [8] features between the real sign and its template and then, by using feature matching and homography estimation, compute an homography to apply to the template of the target sign to paste in order to also reproduce the rotation of the original sign
- it is possible to use GANs to create synthetic signs that look even more realistic when being pasted in a certain context
- GANs can also be used when applying domain transfer to transfer photometric and geometric features from source signs to target signs templates in order to get a target sign that looks as similar as possible to the source sign; for example, we can think of an approach based on CycleGANs or StyleGANs to directly transfer the style from the real sign to the template sign or to move from the domain of templates to the domain of real signs
- it is possible to use other neural networks to predict the position of the signs when being pasted
- it is possible to develop an algorithm that pastes signs accordingly to the actual distribution of the dataset we are augmenting, thus automatically rebalancing the dataset without the need to manually specify a distribution for the pasted signs

Bibliography

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, April 2020. DOI: 10.1007/s10462-020-09825-6. URL: https://doi.org/10.1007%2Fs10462-020-09825-6.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[3] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift, 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://doi.org/10.48550/arxiv.1502.03167.

[4] P. Probst, B. Bischl, and A.-L. Boulesteix. Tunability: importance of hyperparameters of machine learning algorithms, 2018. DOI: 10.48550/ARXIV.1802.09596. URL: https://doi.org/10.48550/arxiv.1802.09596.

[5] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning, 2019. DOI: 10.48550/arxiv.1911.02685. URL: https://doi.org/10.48550/arxiv.1911.02685.

[6] K. He, R. Girshick, and P. Dollár. Rethinking imagenet pre-training, 2018. DOI: 10.48550/ARXIV.1811.08883. URL: https://doi.org/10.48550/arxiv.1811.08883.

[7] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. 1:I–I, 2001. DOI: 10.1109/CVPR.2001.990517. URL: https://doi.org/10.1109/CVPR.2001.990517.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. 1:886–893 vol. 1, 2005. DOI: 10.1109/CVPR.2005.177. URL: https://doi.org/10.1109/CVPR.2005.177.

[10] J. Redmon and A. Farhadi. Yolov3: an incremental improvement, 2018. DOI: 10.48550/arxiv.1804.02767. URL: https://doi.org/10.48550/arxiv.1804.02767.

[11] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: common objects in context, 2014. DOI: 10.48550/arxiv.1405.0312. URL: https://doi.org/10.48550/arxiv.1405.0312.

[12] Object detection. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Object_detection.

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite:3354–3361, 2012. DOI: 10.1109/CVPR.2012.6248074. URL: https://doi.org/10.1109/CVPR.2012.6248074.

[14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013. DOI: 10.48550/arxiv.1311.2524. URL: https://doi.org/10.48550/arxiv.1311.2524.

[16] R. Girshick. Fast r-cnn, 2015. DOI: 10.48550/arxiv.1504.08083. URL: https://doi.org/10.48550/arxiv.1504.08083.

[17] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks, 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://doi.org/10.48550/arxiv.1506.01497.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: single shot MultiBox detector:21–37, 2016. DOI: 10.1007/978-3-319-46448-0_2. URL: https://doi.org/10.1007%2F978-3-319-46448-0_2.

[19] Z. Tian, C. Shen, H. Chen, and T. He. Fcos: fully convolutional one-stage object detection, 2019. DOI: 10.48550/arxiv.1904.01355. URL: https://doi.org/10.48550/arxiv.1904.01355.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection, 2015. DOI: 10.48550/arxiv.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[21] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger, 2016. DOI: 10.48550/arxiv.1612.08242. URL: https://doi.org/10.48550/arxiv.1612.08242.

[22] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: optimal speed and accuracy of object detection, 2020. DOI: 10.48550/arxiv.2004.10934. URL: https://doi.org/10.48550/arxiv.2004.10934.

[23] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, A. V, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, and M. T. Minh. ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference, version v6.1, February 2022. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

[24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection, 2017. DOI: 10.48550/arxiv.1708.02002. URL: https://doi.org/10.48550/arxiv.1708.02002.

[25] M. Tan, R. Pang, and Q. V. Le. Efficientdet: scalable and efficient object detection, 2019. DOI: 10.48550/arxiv.1911.09070. URL: https://doi.org/10.48550/arxiv.1911.09070.

[26] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: efficient convolutional neural networks for mobile vision applications, 2017. DOI: 10.48550/arxiv.1704.04861. URL: https://doi.org/10.48550/arxiv.1704.04861.

[27] S. Kala. Traffic signs recognition: cnn, July 2020. URL: https://medium.com/swlh/traffic-signs-recognition-cnn-ebaa0d18f6ad.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791.

[29] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. A survey of data augmentation approaches for nlp, 2021. DOI: 10.48550/ARXIV.2105.03075. URL: https://doi.org/10.48550/arxiv.2105.03075.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[31] What is image augmentation and how it can improve the performance of deep neural networks. *albumentations.ai*. URL: https://albumentations.ai/docs/introduction/image_augmentation/.

[32] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation, 2017. DOI: 10.48550/ARXIV.1708.04896. URL: https://doi.org/10.48550/arxiv.1708.04896.

[33] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017. DOI: 10.48550/ARXIV.1708.04552. URL: https://doi.org/10.48550/arxiv.1708.04552.

[34] K. K. Singh, H. Yu, A. Sarmasi, G. Pradeep, and Y. J. Lee. Hide-and-seek: a data augmentation technique for weakly-supervised localization and beyond, 2018. DOI: 10.48550/ARXIV.1811.02545. URL: https://doi.org/10.48550/arxiv.1811.02545.

[35] P. Chen, S. Liu, H. Zhao, and J. Jia. Gridmask data augmentation, 2020. DOI: 10.48550/ARXIV.2001.04086. URL: https://doi.org/10.48550/arxiv.2001.04086.

[36] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. Mixup: beyond empirical risk minimization, 2017. DOI: 10.48550/ARXIV.1710.09412. URL: https://doi.org/10.48550/arxiv.1710.09412.

[37] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: regularization strategy to train strong classifiers with localizable features, 2019. DOI: 10.48550/ARXIV.1905.04899. URL: https://doi.org/10.48550/arxiv.1905.04899.

[38] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://doi.org/10.48550/arXiv.1406.2661.

[39] V. Thambawita, P. Salehi, S. A. Sheshkal, S. A. Hicks, H. L. Hammer, S. Parasa, T. d. Lange, P. Halvorsen, and M. A. Riegler. Singan-seg: synthetic training data generation for medical image segmentation. *PLOS ONE*, 17(5):1–24, May 2022. DOI: 10.1371/journal.pone.0267976. URL: https://doi.org/10.1371/journal.pone.0267976.

[40] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:60, 1, July 2019. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[41] L. Taylor and G. Nitschke. Improving deep learning using generic data augmentation, 2017. DOI: 10.48550/ARXIV.1708.06020. URL: https://doi.org/10.48550/arxiv.1708.06020.

[42] H. Naveed. Survey: image mixing and deleting for data augmentation, 2021. DOI: 10.48550/arxiv.2106.07085. URL: https://doi.org/10.48550/arxiv.2106.07085.

[43] Pytorch. URL: https://pytorch.org/.

[44] Tensorflow. URL: https://www.tensorflow.org/.

[45] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation, 2020. DOI: 10.48550/arxiv.2012.07177. URL: https://doi.org/10.48550/arxiv.2012.07177.

[46] Univrses. URL: https://univrses.com/.

[47] J. Wang, Y. Chen, M. Gao, and Z. Dong. Improved yolov5 network for real-time multi-scale traffic sign detection, 2021. DOI: 10.48550/arxiv.2112.08782. URL: https://doi.org/10.48550/arxiv.2112.08782.

[48] Traffic sign detection via improved sparse r-cnn for autonomous vehicles. *Journal of Advanced Transportation*, March 2022. ISSN: 0197-6729. DOI: 10.1155/2022/3825532. URL: https://doi.org/10.1155/2022/3825532.

[49] D. Tabernik and D. Skočaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1427–1440, 2020. DOI: 10.1109/TITS.2019.2913588. URL: https://doi.org/10.1109/TITS.2019.2913588.

[50] Y.-K. Park, H. Park, Y.-S. Woo, I.-G. Choi, and S.-S. Han. Traffic landmark matching framework for hd-map update: dataset training case study. *Electronics*, 11(6), 2022. ISSN: 2079-9292. DOI: 10.3390/electronics11060863. URL: https://doi.org/10.3390/electronics11060863.

[51] K. Singh and N. Malik. Cnn based approach for traffic sign recognition system. *Advanced Journal of Graduate Research*, 11(1):23–33, September 2021. DOI: 10.21467/ajgr.11.1.23-33. URL: https://journals.aijr.org/index.php/ajgr/article/view/3851.

[52] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks, 2018. DOI: 10.48550/ARXIV.1812.04948. URL: https://doi.org/10.48550/arxiv.1812.04948.

[53] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017. DOI: 10.48550/ARXIV.1703.10593. URL: https://doi.org/10.48550/arxiv.1703.10593.

[54] D. Horn and S. Houben. Fully automated traffic sign substitution in real-world images for large-scale data augmentation:465–471, 2020. DOI: 10.1109/IV47402.2020.9304547. URL: https://doi.org/10.1109/IV47402.2020.9304547.

[55] A. Konushin, B. Faizov, and V. Shakhuro. Road images augmentation with synthetic traffic signs using neural networks, 2021. DOI: 10.48550/arxiv.2101.04927. URL: https://doi.org/10.48550/arxiv.2101.04927.

[56] L. Tabelini, R. Berriel, T. M. Paixão, A. F. De Souza, C. Badue, N. Sebe, and T. Oliveira-Santos. Deep traffic sign detection and recognition without target domain real images, 2020. DOI: 10.48550/ARXIV.2008.00962. URL: https://doi.org/10.48550/arxiv.2008.00962.

[57] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf:2564–2571, 2011. DOI: 10.1109/ICCV.2011.6126544. URL: https://doi.org/10.1109/ICCV.2011.6126544.

[58] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[59] N. Soufi and M. Valdenegro-Toro. Data augmentation with symbolic-to-real image translation gans for traffic sign recognition, 2019. DOI: 10.48550/arxiv.1907.12902. URL: https://doi.org/10.48550/arxiv.1907.12902.

[60] Hydra. URL: https://hydra.cc.

[61] Road signs in sweden. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Road_signs_in_Sweden.

[62] Road signs in sweden. *Wikimedia*. URL: https://commons.wikimedia.org/wiki/Road_signs_in_Sweden.

[63] Comparison of european road signs. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Comparison_of_European_road_signs.