Roto-translation Equivariant YOLO for Aerial Images

Benjamin Maurel BENJAMIN.MAUREL@ENSAE.FR
[*] *Thales, ENSAE/ENS Paris-Saclay, France*

Samy Blusseau SAMY.BLUSSEAU@MINESPARIS.PSL.EU
Santiago Velasco-Forero SANTIAGO.VELASCO@MINES-PARISTECH.FR
Mines Paris, PSL University, France

Teodora Petrisor TEODORA.PETRISOR@THALESGROUP.COM
Thales, Research and Technology in France[†]

Abstract

This work introduces Eq-YOLO, an Equivariant One-Stage Object Detector based on YOLO-v8 incorporating group convolutions to handle rotational transformations. We show the interest of using equivariant-transforms to improve the detection performance on rotated data over the regular YOLO-v8 model while dividing the number of parameters to train by a factor greater than three.

Keywords: Object Detection, CNN, YOLO, Equivariance, Symmetry

1. Introduction

Object detection in aerial images is a challenging task in particular due to the size of the object of interest (down to only a few pixels), the number of objects to detect and their variability in terms of orientation. This inherently demands equivariance to a range of transformations such as translations, scaling or rotations. Indeed, detecting objects from various angles is essential, requiring not only class consistency (invariance) but also adjusted bounding boxes according to their orientation (equivariance). Designing architectures which are equivariant and invariant to the symmetries of the data allows to meet the detection targets while reducing the necessary amount of parameters and training samples Cohen and Welling (2016); Han et al. (2021). This paper focuses on making the YOLO-v8 Redmon et al. (2016b); Jocher et al. (2023) architecture equivariant to rotations. Through the incorporation of group-convolutions to handle rotational transformations, our proposed Eq-YOLO achieves improved performance and robustness, all while notably reducing the overall number of parameters.

2. Rotational Equivariance

Our object detection model can be defined as follows: Given an input signal $f : \mathbb{R}^2 \to \mathbb{R}$, the model Φ is defined such that $\Phi : f \mapsto \Phi(f) \in (\mathbb{Z}^4 \times [0,1]^{n_c})^{n_{obs}}$, where n_c is the number of classes and n_{obs} the number of detections.

For a given input signal f, $\Phi(f)$ provides the coordinates of the bounding boxes of the fixed n_{obs} detections along with the probabilities for each of the n_c classes. Our goal is to achieve equivariance for bounding box predictions and invariance for class predictions.

[*] This work has been done during a research internship at Thales, Research and Technology in France
[†] This work has been partly funded by the EU Grant 101056885

An operator $\phi : X \to Y$ is said to be *G-equivariant* for a given group G if there exist two *actions* π and π' on X and Y, respectively, such that:

$$\forall g \in G, \forall x \in X, \phi(\pi_g(x)) = \pi'_g(\phi(x)) \tag{1}$$

If $\forall g \in G, \forall x \in X, \phi(\pi_g(x)) = \phi(x)$ the operator is *invariant.*

2.1. Group-convolution as an equivariant version of the convolution for discrete groups

In our case, to achieve rotational and translational equivariance, we focus on $G = \mathbb{Z}^2 \times C_8$, where C_8 represents the group of rotations by angles multiples of $\pi/4$ radians (45°), a discrete and finite subgroup of $SO(2)$.

Similar to how convolutional neural networks (CNNs) use cross-correlation operations for translation equivariance, we adopt Group-convolution as introduced in Weiler et al. (2018); Hoogeboom et al. (2018); Bekkers et al. (2018) to extend this property to rotations. This approach provides equivariance to the chosen group. In our case, this gives us, for a given signal $\tilde{f} : G \to \mathbb{Z}$ and a kernel $\tilde{k} : G \to \mathbb{Z}$:

$$[\tilde{k} \star_G \tilde{f}](y, g_j) := \sum_{g_n \in C_8} \sum_{x \in \mathbb{Z}^2} \tilde{k}(g_j^{-1}(x - y), g_j^{-1} g_n) \tilde{f}(x, g_n) = \sum_{g_n \in C_8} [g_j \cdot k_{g_n} \star f_{g_n}](y, g_j) \tag{2}$$

with $k_{g_n} = \tilde{k}(\cdot, g_n) : \mathbb{Z}^2 \to \mathbb{Z}$, and $f_{g_n} = \tilde{f}(\cdot, g_n) : \mathbb{Z}^2 \to \mathbb{Z}$.

Notably, the signal $\tilde{f}$ operates over $\mathbb{Z}^2 \times C_8$, different from our initial signal. Before applying group convolutions, we need to 'lift' the signal by stacking rotated versions. Hence, for the initial group-convolution, $\tilde{f} = (g_n \cdot f)_{g_n \in C_8}$. Features remain in this lifted space until `projection` back to the original space before the final output.

Importantly, this expression enables efficient implementation of group convolutions using only native `conv2D` functions available in popular deep learning frameworks. The f_{g_n} signals are stacked into an 8-channel input, followed by a conventional convolution using the $g_j \cdot k_{g_n}$ kernel. This integration ensures seamless application of group convolutions, with input and output feature maps accommodating 8-channels. The first row remains learnable, subsequent rows correspond to transformed versions influenced by the actions of distinct group elements $g_n \in C_8$.

3. Eq-`YOLO`

`YOLO`-v8 Jocher et al. (2023) is an advanced single-stage object detection model with a fully convolutional neural network backbone (more details in Appendix B). One of the particularities of `YOLO`-v8 is to predict separately the class and the bounding-box. The essence of Eq-`YOLO` resides in the transformation of all internal operations using equivariant operations[1]. Each layer of `YOLO`-v8 must be replaced by its equivariant counter-part. We implemented the equivariant version of `Conv`, `C2f`, `SPPF`, `Upsample`, `Concatenation` and `Detection Head` layers using the `ESCNN` library Cesa et al. (2022). As done in Weiler and

1. For this preliminary work we only considered enforcing equivariance to discrete rotations

Cesa (2019) or Cesa et al. (2022), to compete fairly with `YOLO`-v8 we construct our Eq-`YOLO` model with the same number of channels through the entire network except for the `Detection Head`.

Indeed, in the `Detection Head`, we need to add a projection before the final output to avoid changing the output shape. The objective is to project $\tilde{f} : G \to \mathbb{R}$ onto a signal $f : \mathbb{Z}^2 \to \mathbb{R}$.

3.1. Projection layer for the class

For class prediction we can directly integrate over C_8. Moreover, if we denote as $f : X \to Y$ the prediction of the class after projection, f is invariant. Then, for any function $\Psi : Y \to Y$, $\Psi \circ f$ is also invariant. Since no matter when we project into the space of output, we keep the invariance property, applying the projection from the start of the `Detection Head` allows to keep the same computational cost as `YOLO`-v8.

3.2. Projection layer for the bounding box

As in Li et al. (2023a) and Tian et al. (2019), the network predicts fixed-point-centered bounding boxes. The network output represents the distances between the fixed point and the four sides of the box, denoted as (l, t, r, b) for left, top, right, and bottom, respectively. We denote as $f_\eta(x)$ a box predicted by the network[2].

Knowing how G acts on the output space allows to use all the information given by the network. For easier representation, we consider $G = C_4$[3], but this can be generalised to $G = C_8$. We seek a projection Ψ such that $\Psi(g \cdot f_\eta(x)) = g \cdot (l, t, b, r)$. Regardless of the layer, the action of g on features is to shift channels over C_4.

$$g_1 \cdot f_\eta(x) = g_1 \cdot \begin{bmatrix} l_{g_0} & t_{g_0} & b_{g_0} & r_{g_0} \\ l_{g_1} & t_{g_1} & b_{g_1} & r_{g_1} \\ l_{g_2} & t_{g_2} & b_{g_2} & r_{g_2} \\ l_{g_3} & t_{g_3} & b_{g_3} & r_{g_3} \end{bmatrix} = \begin{bmatrix} l_{g_1} & t_{g_1} & b_{g_1} & r_{g_1} \\ l_{g_2} & t_{g_2} & b_{g_2} & r_{g_2} \\ l_{g_3} & t_{g_3} & b_{g_3} & r_{g_3} \\ l_{g_0} & t_{g_0} & b_{g_0} & r_{g_0} \end{bmatrix} \tag{3}$$

Hence, we can find several correct ways to define such a Ψ. The easiest way is to define it such that:

$$\Psi^1_{init} : f_\eta(x) \to \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot f_\eta(x) = \begin{bmatrix} l_{g_0} & t_{g_0} & b_{g_0} & r_{g_0} \end{bmatrix}$$

However, this approach discards valuable information propagated in the network. Alternatively, we could consider:

$$\Psi^2_{init} : f_\eta(x) \to f_\eta(x) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T = \begin{bmatrix} l_{g_0} & l_{g_1} & l_{g_2} & l_{g_3} \end{bmatrix}^T$$

Again, this approach under-utilizes the information.

To use all the information provided by our network, we want to leverage the fact that $l_{g_0}, r_{g_1}, b_{g_2}, t_{g_3}$ are four different estimators of l. With $\Psi = (\Psi^l, \Psi^t, \Psi^r, \Psi^b)$ we set:

2. $f_\eta(x)$ is a feature map of shape $\mathbb{R}^{|G| \times 4}$.
3. where the action g on a prediction (l, t, b, r) is a cyclic shift, e.g., $g_1 \cdot (l, t, b, r) = (t, b, r, l)$

For l, we have: $\Psi^l(f_\eta(x)) = \frac{1}{|C_4|}\sum_{i=0}^{3} f_\eta^{i,(4-i) \bmod 4}(x)$.

For t, we have: $\Psi^t(f_\eta(x)) = \frac{1}{|C_4|}\sum_{i=0}^{3} f_\eta^{i,(5-i) \bmod 4}(x)$.

For r, we have: $\Psi^r(f_\eta(x)) = \frac{1}{|C_4|}\sum_{i=0}^{3} f_\eta^{i,(6-i) \bmod 4}(x)$.

For b, we have: $\Psi^b(f_\eta(x)) = \frac{1}{|C_4|}\sum_{i=0}^{3} f_\eta^{i,(3-i) \bmod 4}(x)$.

By combining the estimators of our targeted bounding box we leverage all the information given by our network and hope to improve the accuracy of the model[4].

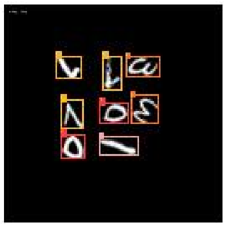
4. Experiments and Analysis

We first created a specific object-detection dataset based on MNIST named `OD-MNIST` of images featuring randomly positioned MNIST digits within a 134x134 canvas. While for the *train set*, the digits are NOT rotated, the *test set* includes rotated numbers to assess model robustness. An illustrative example is provided in Figure 1(*b*).

For comparison against the nano version of `YOLO`-v8, we introduced our proposed models, Eq-`YOLO`-v1 and Eq-`YOLO`-v2[5], which have significantly fewer parameters. Both of our models surpassed `YOLO`-v8's performance in the considered metrics (Figure 1(*a*)[6]).

Model	Size	Precision	Recall	mAP-50
Eq-`YOLO`-v2	0.4	0.92 +/- .03	0.91 +/- .06	0.93 +/- .04
Eq-`YOLO`-v1	1.1	0.89 +/- .03	0.86 +/- .02	0.93 +/- .01
`YOLO`-v8	3.4	0.82 +/- .01	0.84 +/- .02	0.89 +/- .01

(*a*) Performance Metrics for different models (Precision and Recall for IoU threshold of 0.5 (Everingham et al. (2010) part 2.4.3) Sizes in Millions of Parameters. mAP stands for mean Average Precision



(*b*) Image from `OD-MNIST` test set

Figure 1: Results on the `OD-MNIST` dataset.

Secondly, we tested Eq-`YOLO` on aerial images. We have used xView Lam et al. (2018) restricted to four classes for training and an open-source drone-view dataset Aerial-Cars-Dataset (2022) for testing (an example image is given in Appendix A). This test dataset has been chosen to simultaneously verify the generalization capabilities of our model. At this stage we only restricted our evaluation to the most representative classes in the test set: `car` and `truck`.

4. When using C_8 instead of C_4, a similar idea applies, as $C_4 \subset C_8$, for example for l, it becomes: $\Psi^l(f_\eta(x)) = \frac{2}{|C_8|}\sum_{i=0}^{3} f_\eta^{2\times i, 2(4-i) \bmod 8}(x)$.
5. Eq-`YOLO`-v2 uses the last version of the projection layer. For the class prediction, the projection into the output space is done earlier which induces a reduction of the number of parameters.
6. The uncertainty is computed by training ten times the model with random initialisation. All models are trained with the same data augmentation, random rotation of ±90 deg, mirror up/down, left/right are applied.

In Table 1, we show the results for Eq-`YOLO` as compared to the baseline `YOLO`-v8 model for each of the two classes: `car` and `truck`. The size of the model is given in number of weights at training (in Millions) and the inference time was estimated on CPU only.

Category	Model	AP-50	Precision	Recall	Size	CPU Inference Time
Cars	Eq-`YOLO`	0.75	0.77	0.81	$2.8M$	252 ms
	`YOLO`-v8	0.66	0.79	0.66	$12M$	149 ms
Trucks	Eq-`YOLO`	0.32	0.38	0.34	$2.8M$	252 ms
	`YOLO`-v8	0.15	0.21	0.31	$12M$	149 ms

Table 1: Performance metrics on aerial data.

The Average Precision of Eq-`YOLO` is better by a large margin on each class, while the model size is reduced by a factor of 4.8. On the other hand, the unfavorable difference in inference time on CPU should be tackled with further work on the projection layer.

Limitations

The main limitation of Equivariant-YOLO is its time inference.

On Appendix 5, we can see the inference-time and number of operations (Flops) estimated for the two models. These figures has been obtained on CPU and the average-time is a mean over 20 measures. If the inference-time is comparable in the other layers, the last layer, corresponding to the head module cannot compete with the `YOLO`-v8. That is why we need to project the class back to $\mathbb{R}^2$ as quickly as possible. By projecting from the start of the class prediction head instead of at the end, we can reduce the inference-time difference by 25%.

Other directions for improving the inference time are to act on the box prediction head. Indeed, if we had oriented bounded boxes, the box prediction head will then predict l, t, r, b, θ for each of the "center points". This $\theta \in [0, 360]$ is the angle that orients the box. Note that by choosing $\theta \in [0, 360]$, we have l, t, r, b **which become invariant**. This allows us to project those values early on (as for the class) and hence expect important gains in inference time.

Future Directions

Our work lays the path for further advancements in equivariant object detection in aerial images. There are several promising avenues for further investigation, including the exploration of rotated bounding boxes, i.e with sides not necessarily parallel to the image axis. This could lead to both enhancements in object localization and faster prediction, as the representation of bounding boxes as (l, t, r, b) would become invariant in the process.

References

Aerial-Cars-Dataset. Aerial cars dataset. https://universe.roboflow.com/pian-jiangfeng-gmail-com/aerial-cars, jul 2022. URL https://universe.roboflow.com/pian-jiangfeng-gmail-com/aerial-cars. visited on 2023-10-04.

Erik J. Bekkers, Maxime W. Lafarge, Mitko Veta, Koen A. J. Eppenhof, Josien P. W. Pluim, and Remco Duits. Roto-translation covariant convolutional networks for medical image analysis. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 440–448, Cham, 2018. Springer International Publishing.

Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build E(N)-equivariant steerable CNNs. In *International Conference on Learning Representations*, 2022.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

Jian Ding, Nan Xue, Yang Long, Gui-Song Xia, and Qikai Lu. Learning roi transformer for detecting oriented objects in aerial images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

Jiaming Han, Jian Ding, Nan Xue, and Gui-Song Xia. Redet: A rotation-equivariant detector for aerial object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2786–2795, 2021.

Emiel Hoogeboom, Jorn W.T. Peters, Taco S. Cohen, and Max Welling. Hexaconv. In *International Conference on Learning Representations*, 2018.

Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Software: YOLO by Ultralytics, version 8.0.0., January 2023.

Darius Lam, Richard Kuzma, Kevin McGee, Samuel Dooley, Michael Laielli, Matthew Klaric, Yaroslav Bulatov, and Brendan McCord. xview: Objects in context in overhead imagery, 2018.

Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. Generalized focal loss: Towards efficient representation learning for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3139–3153, 2023a. doi: 10.1109/TPAMI.2022.3180392.

Yuxuan Li, Qibin Hou, Zhaohui Zheng, Mingming Cheng, Jian Yang, and Xiang Li. Large selective kernel network for remote sensing object detection. *ArXiv*, 2023b.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

Delond Angelo Jimenez Nixon. Computer vision neural network using YOLOv4 for underwater fish video detection in Roatan, Honduras. In *2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*, pages 1–5. IEEE, 2021.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016a.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016b.

Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection, 2019.

Ultralytics. Ultralytics Repository. https://github.com/ultralytics/ultralytics, 2023.

Maurice Weiler and Gabriele Cesa. General E(2)-equivariant steerable cnns. *Advances in neural information processing systems*, 32, 2019.

Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant CNNs. In *CVPR*, 2018.

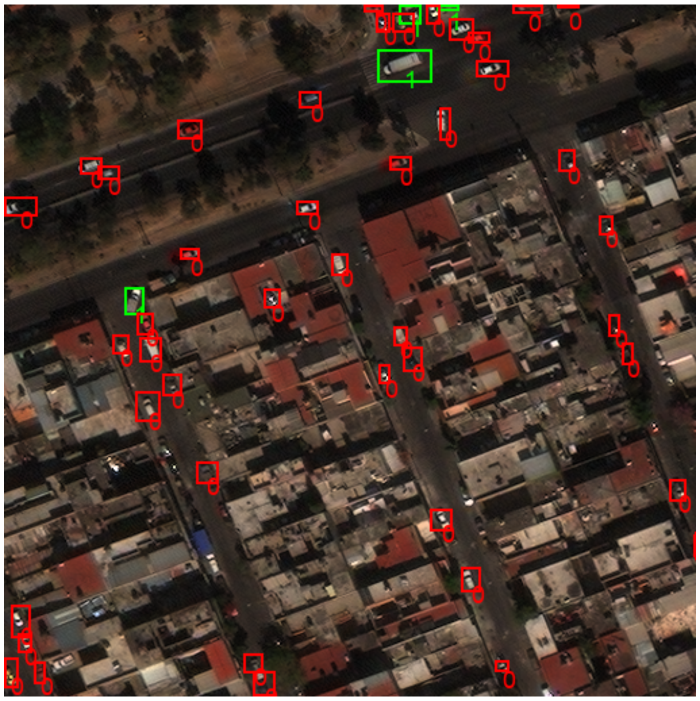


Figure 2: Illustration of the object detection task on xView Dataset. Cars are in red and correspond to class 0, Trucks are in green and correspond to class 1.

Appendix A. Object Detection

Generic object detection aims at locating and classifying objects in an image. An object detection model is usually a deep learning model that locates and classify objects by predicting a rectangular box around them as well as an associated class. There have been two main approaches for solving this problem in the literature: two-stage methods and single-stage ones. The two-stage approaches Girshick (2015) follow a traditional object detection pipeline, first generating region proposals and then classifying each proposal into different object categories. One stage detectors Redmon et al. (2016a), view object detection as a regression or classification problem, adopting a unified framework to achieve categories and locations simultaneously.

One of the things that makes the task of object detection significantly more complex than classification or even generation tasks is the absence of prior knowledge about the number of objects to detect, as opposed to classification. This detail may seem trivial, but it has a profound impact. The output of an object detection network is constantly fixed, as it depends on the fixed architecture itself. The challenge is then to detect a variable number of objects in the image with a fixed number of outputs.

A.1. Two-Stage Approaches

One perspective is to consider that while an image may contain multiple objects, one can always frame this as a problem involving a single object. Initially, regions of interest (RoIs Girshick et al. (2014); Girshick (2015)) are identified where only one object needs to be detected. Then, a neural network, denoted as f, is applied to this RoI to extract the bounding box and its associated class. The advantage of this approach is that well-established methods exist for extracting RoIs Girshick et al. (2014); Girshick (2015). However, the downside is that if there are n objects in the image, this method requires applying n times f.

Two-stage object detection methods, such as Faster R-CNN Girshick (2015) and R-CNN Girshick et al. (2013), have been influential in achieving state-of-the-art results. The two stages are as follows:

1. Region Proposal Generation: In the first stage, a region proposal network (RPN) is used to generate a set of candidate object regions (bounding boxes) within the image. These regions are potential locations where objects may be present. RPNs use anchor boxes and regression techniques to propose regions efficiently.

2. Object Classification and Refinement: In the second stage, the proposed regions are fed into a classifier (e.g., a convolutional neural network) to determine the class labels and refine the bounding box coordinates of the detected objects. Post-processing steps can be applied to filter out duplicate detections and improve localization accuracy.

These methods have several advantages:

- High Accuracy: They often achieve higher detection accuracy due to the explicit region proposal step, which focuses computation on potential object regions.

- Precise Localization: The two-stage architecture allows for precise localization of objects, making them suitable for tasks where accurate object boundaries are crucial.

Today, these types of algorithms achieve state-of-the-art performance in term of accuracy Ding et al. (2019); Li et al. (2023b), however, they also exhibit some limitations with respect to the second class of approaches:

- Slow inference: The multi-stage process can result in slower inference times, making them less suitable for real-time applications.

- Increased complexity: Implementing and training two-stage models can be more complex and resource-intensive.

A.2. One-Stage Approaches

On the other hand, the alternative approach, considered also in Eq-`YOLO`, takes a completely different direction. Instead of detecting one object at a time, a fixed number, denoted as $n_{\max}$, of detections is made in each pass through the network, where $n_{\max}$ is significantly greater than the number of objects in the dataset. At the end of the process only the "best detections" are retained.

One-stage object detection methods, exemplified by models like YOLO (You Only Look Once) Redmon et al. (2016a) and SSD (Single Shot MultiBox Detector) Liu et al. (2016), have gained popularity for their efficiency and speed. They operate in a single pass through the network and directly predict object bounding boxes and class labels.
Key advantages of one-stage methods include:

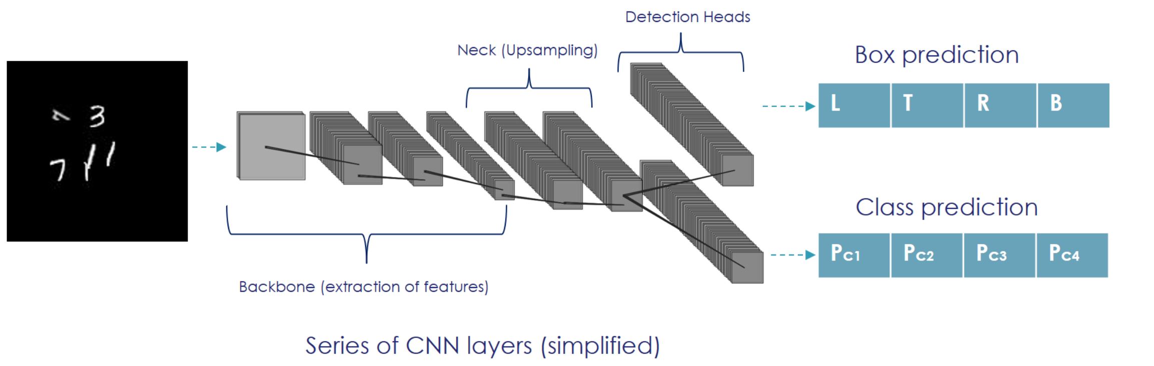
1. Single-Pass Prediction: These methods predict object classes and bounding boxes in a single forward pass through the network, making them faster than two-stage approaches.

2. Efficiency: One-stage methods are well-suited for real-time and embedded applications due to their speed.

In this paper we focused on YOLO models and more precisely on the very recent YOLO-v8 Ultralytics (2023), which we detail in the following.

Appendix B. Overview of YOLO-v8 model architecture

The YOLO-v8 model architecture consists of three main components, illustrated in Figure 3. The "backbone" network exhibits the characteristic structure of convolutional networks, it is based on architectures such as ResNet or DarkNet (Lin et al. (2017); Targ et al. (2016); Nixon (2021). It is composed entirely of convolutional layers and aims to extract meaningful features from the images. Following the backbone is the "neck" which, in addition to convolutional layers, also incorporates "UpSampling" layers to transform the features extracted by the backbone back into the desired output shape. The third part is composed of "head modules." Features are extracted from three different locations within the network to make final predictions.



$$\boxed{\text{Given an input signal } f : R^2 \rightarrow R, \text{ our model } \Phi \text{ is defined such that } \Phi : f \rightarrow (Z^4 \times [0,1]^{n_c} \quad)^{n_{obs}}}$$

Figure 3: A simplified view of YOLO-v8

B.1. Backbone

The backbone of the YOLO-v8 model serves as the foundation for feature extraction from input images. It typically follows a structure reminiscent of convolutional neural networks

(CNNs) like ResNet Lin et al. (2017); Targ et al. (2016). The primary purpose of the backbone is to transform raw image data into higher-level feature representations, which can be used for subsequent object detection tasks.

The backbone consists of multiple convolutional layers organized in a deep hierarchical manner. These layers capture features at different scales, starting from low-level details such as edges and textures and gradually moving towards more abstract and high-level features that are indicative of objects and their spatial relationships.

The choice of the specific architecture for the backbone can have a significant impact on the performance of the YOLO-v8 model. Common choices include ResNet, Darknet, or custom-designed architectures tailored to the requirements of the object detection task.

B.2. Neck

The neck of the YOLO-v8 model is an intermediate component that bridges the gap between the feature maps extracted by the backbone and the final output predictions. While the backbone focuses on feature extraction, the neck module is responsible for shaping these features into a format suitable for object detection.

In addition to convolutional layers, the neck often incorporates “UpSampling” layers, which increase the spatial resolution of feature maps. This is crucial because object detection requires precise localization, and higher-resolution feature maps are better suited for this purpose.

The neck module plays a crucial role in aggregating multi-scale features from different levels of the backbone. It helps the model gather information from both fine-grained and coarse-grained feature maps, enabling it to detect objects of varying sizes and complexities.

B.3. Head Modules

The head modules of the YOLO-v8 model are responsible for making final predictions about the objects detected in an image. These head modules typically operate on feature maps obtained from different locations within the “neck”, allowing them to capture context and details at multiple scales. More precisely, there are two “UpSampling” layers which implies three different sizes of features.

YOLO-v8 employs a multi-head architecture, meaning that it uses multiple sets of prediction layers, often one for each scale of the feature maps. In our case, this represents three different scales. Each set of prediction layers consists of convolutional and fully connected layers that output predictions for object bounding boxes and their associated class probabilities.

The multi-head approach is essential for the YOLO-v8 model to detect objects at various scales effectively. By considering features from different parts of the network, it can handle both small and large objects within the same image.

B.4. YOLO-v8 architecture: Summary

In summary, the YOLO-v8 model backbone extracts features from input images, the neck module prepares these features for object detection, and the head modules make final predictions about the objects present in the image. This modular architecture enables YOLO-v8 to perform accurate and efficient object detection across a wide range of scenarios and object sizes.

Appendix C. Details on Equivariance

C.1. Group Theory

Definition 1 (Group) *A group is a pair* $(G, \cdot)$ *containing a set* G *and a binary operation* $\cdot : G \times G \to G$, $(h, g) \mapsto h \cdot g$, *satisfying the following group axioms:*

- ***Associativity:*** $\forall a, b, c \in G, \quad a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- ***Identity:*** $\exists e \in G, \forall g \in G, \quad g \cdot e = e \cdot g = g$
- ***Inverse:*** $\forall g \in G, \exists g^{-1} \in G, \quad g \cdot g^{-1} = g^{-1} \cdot g = e$

The binary operation $\cdot$ is called the group law. It can be proven that the inverse g^{-1} of an element g is unique.

Example (Cyclic Group) The set of all the complex n-th roots of unity $\{e^{ik2\pi/n} \,|\, 0 \leq k < n\}$ forms a group under multiplication. This group is isomorphic to the group $\mathbb{Z}/n\mathbb{Z}$. Indeed, we can define a homomorphism f between them as

$$f : e^{ik2\pi/n} \mapsto k.$$

One can verify that this is indeed an isomorphism.

This group is also called the Cyclic Group of order n, often indicated as C_n. More abstractly, this group can be defined as

$$C_n = \{e = g^0, g^1, g^2, \ldots, g^{n-1} \,|\, g^i = g^j \Leftrightarrow i \equiv j \pmod{n}\}.$$

Note that any element of the group can be identified by a power of the generating element g. We can also notice that applying an element of C_n onto the complete set of element of C_n shifts the whole set: $g^3 \cdot \{g^0, g^1, ..., g^{n-1}\} = \{g^3, g^4, ..., g^2\}$.
We will sometimes refers to the shift as a "rotation" of the elements of C_n since it is a cyclic group.

Definition 2 (Group Action and G-Space) *Given a group* G*, a (left)* G*-space* X *is a set* X *equipped with a group action* $G \times X \to X$, $(g, x) \mapsto g \cdot x$*, i.e., a map satisfying the following axioms:*

- ***Identity:*** $\forall x \in X, \quad e \cdot x = x$
- ***Compatibility:*** $\forall a, b \in G, \forall x \in X, \quad a \cdot (b \cdot x) = (ab) \cdot x$

In this case, G *is said to act on* X*.*

C.2. Translation and Rotation Equivariance

Definition 3 (Invariance and Equivariance) *Given a group G and two G-sets X and Y defined by their respective group action such that:*
$G \times X \to X, (g, x) \to g \cdot x$
$G \times Y \to Y, (g, y) \to g * y$

- *a function $f : X \to Y$ is said to be equivariant if for all $x \in X$ and for all $g \in G$, we have $f(g \cdot x) = g * f(x)$.*
- *a function $f : X \to Y$ is said to be invariant if for all $x \in X$ and for all $g \in G$, we have $f(g \cdot x) = f(x)$.*

As highlighted in Figure 4, it is important to notice here that the action of G on X can be really different than the action of G on Y: "rotating" a digit is not necessarily the same thing as "rotating" a detection.

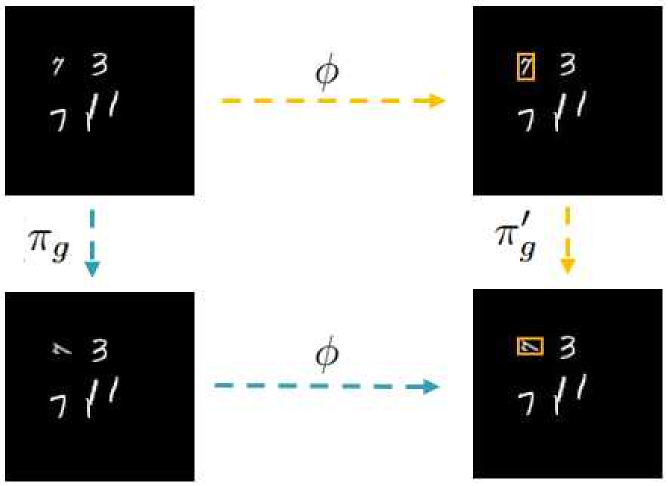
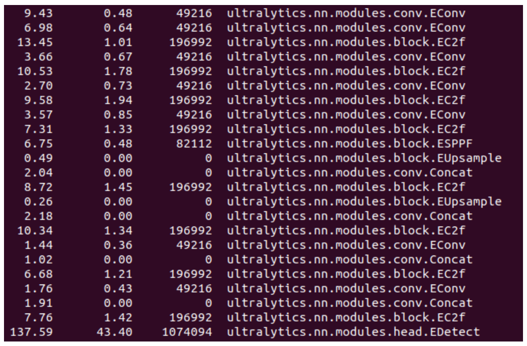


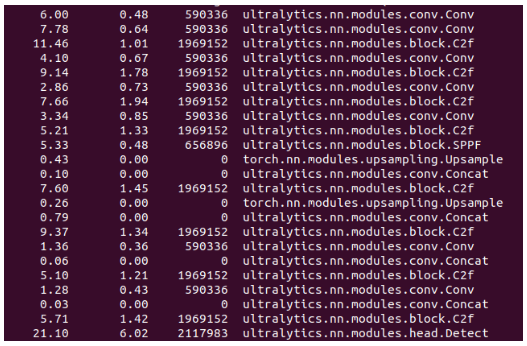
Figure 4: Diagram illustrating the equivariance property, π_g and π'_g represent the actions of the group on different spaces. We obtain the same output whether we first apply the model and then apply the group action π'_g, or if we apply the group action π_g and the apply the model.

Appendix D. Comparaison of time inference at each layer between Eq-YOLO and YOLO-v8



```
   9.43       0.48      49216  ultralytics.nn.modules.conv.EConv
   6.98       0.64      49216  ultralytics.nn.modules.conv.EConv
  13.45       1.01     196992  ultralytics.nn.modules.block.EC2f
   3.66       0.67      49216  ultralytics.nn.modules.conv.EConv
  10.53       1.78     196992  ultralytics.nn.modules.block.EC2f
   2.70       0.73      49216  ultralytics.nn.modules.conv.EConv
   9.58       1.94     196992  ultralytics.nn.modules.block.EC2f
   3.57       0.85      49216  ultralytics.nn.modules.conv.EConv
   7.31       1.33     196992  ultralytics.nn.modules.block.EC2f
   6.75       0.48      82112  ultralytics.nn.modules.block.ESPPF
   0.49       0.00          0  ultralytics.nn.modules.block.EUpsample
   2.04       0.00          0  ultralytics.nn.modules.conv.Concat
   8.72       1.45     196992  ultralytics.nn.modules.block.EC2f
   0.26       0.00          0  ultralytics.nn.modules.block.EUpsample
   2.18       0.00          0  ultralytics.nn.modules.conv.Concat
  10.34       1.34     196992  ultralytics.nn.modules.block.EC2f
   1.44       0.36      49216  ultralytics.nn.modules.conv.EConv
   1.02       0.00          0  ultralytics.nn.modules.conv.Concat
   6.68       1.21     196992  ultralytics.nn.modules.block.EC2f
   1.76       0.43      49216  ultralytics.nn.modules.conv.EConv
   1.91       0.00          0  ultralytics.nn.modules.conv.Concat
   7.76       1.42     196992  ultralytics.nn.modules.block.EC2f
 137.59      43.40    1074094  ultralytics.nn.modules.head.EDetect
```

(*a*) Eq-YOLO



```
   6.00       0.48     590336  ultralytics.nn.modules.conv.Conv
   7.78       0.64     590336  ultralytics.nn.modules.conv.Conv
  11.46       1.01    1969152  ultralytics.nn.modules.block.C2f
   4.10       0.67     590336  ultralytics.nn.modules.conv.Conv
   9.14       1.78    1969152  ultralytics.nn.modules.block.C2f
   2.86       0.73     590336  ultralytics.nn.modules.conv.Conv
   7.66       1.94    1969152  ultralytics.nn.modules.block.C2f
   3.34       0.85     590336  ultralytics.nn.modules.conv.Conv
   5.21       1.33    1969152  ultralytics.nn.modules.block.C2f
   5.33       0.48     656896  ultralytics.nn.modules.block.SPPF
   0.43       0.00          0  torch.nn.modules.upsampling.Upsample
   0.10       0.00          0  ultralytics.nn.modules.conv.Concat
   7.60       1.45    1969152  ultralytics.nn.modules.block.C2f
   0.26       0.00          0  torch.nn.modules.upsampling.Upsample
   0.79       0.00          0  ultralytics.nn.modules.conv.Concat
   9.37       1.34    1969152  ultralytics.nn.modules.block.C2f
   1.36       0.36     590336  ultralytics.nn.modules.conv.Conv
   0.06       0.00          0  ultralytics.nn.modules.conv.Concat
   5.10       1.21    1969152  ultralytics.nn.modules.block.C2f
   1.28       0.43     590336  ultralytics.nn.modules.conv.Conv
   0.03       0.00          0  ultralytics.nn.modules.conv.Concat
   5.71       1.42    1969152  ultralytics.nn.modules.block.C2f
  21.10       6.02    2117983  ultralytics.nn.modules.head.Detect
```

(*b*) YOLO-v8

Figure 5: Inference Time (ms), Estimated Flops (GFlops), Number of parameters and Type of layer. Computed on CPU, values are indicative for comparison only.