Auto-Augmented Transparent Cage Images for Improved Bird Detection in Cages

1st Shayan Mousavinia
Iran University of Science and Technology
sh_mousavinia@comp.iust.ac.ir

2nd Mohammad Reza Mohammadi
Iran University of Science and Technology
mrmohammadi@ee.sharif.edu

I. Abstract

An automated method for detecting birds is prevalent, however it is difficult to use when the birds are kept in cages. Although there are numerous datasets that may be used to train models for bird detection, including COCO, Birdsnap, and Birds-450, producing a dataset with caged birds can be time-consuming and expensive. In this article, we provide a technique to increase the accuracy of bird detection inside cages without the necessity for a specific dataset.

In order to evaluate the performance of models built solely from the COCO base with models built from COCO plus the cage, we added a transparent cage image to the COCO data set. We have numerous places, though, that these tasks don't. In this research, we investigate the impact of including a cage on the objects that we are using to train our model. Our findings indicate that adding clear cages to our birds has a significant impact on testing accuracy and mAP. Our mAP50 value was approximately 20% after the original test without utilizing this approach, and after using it, we reached a mAP50 value of 61%.

II. Introduction

In recent years, Convolutional Neural Networks (ConvNets) have achieved higher accuracies in tasks such as object detection and classification, thanks to increasing hardware power and larger labeled datasets. Data augmentation techniques have also been widely used to improve the generalization of deep learning models [1]–[5] and reduce overfitting, with common methods including rotation, flipping, cropping, and adding noise. We focus on identifying birds inside a cage, where the cage greatly impacts the image and makes it difficult to build accurate models using the COCO dataset. Collecting data for this task is a careful and time-consuming process. While basic augmentation methods can make models insensitive to noise and rotation, we propose a new augmentation method that makes our model sensitive to the cage. Our goal is to improve the accuracy of existing algorithms like Yolo, which struggle with cage bird detection due to the cage's impact on the image. By using our novel approach, we aim to increase the accuracy of bird detection in cages and address the limitations of existing algorithms.

- Synthetic Dataset:
  The process of generating new label data can be both costly and time-consuming, but synthetic datasets offer a promising alternative. Anantrasirichai et al. utilized a synthetic dataset [6] to detect surface deformation that exhibited a strong statistical correlation to volcanic eruptions. Their study demonstrated that training with artificial samples can improve the ability of convolutional neural networks (CNNs) to detect volcano deformation in satellite images. Additionally, they proposed an efficient workflow for developing automated systems and found that CNNs trained with synthetic data achieved better performance than those trained with real data. This technique of generating synthetic data has numerous applications beyond volcanic activity detection. In many critical scenarios, it is imperative to publish personal data while ensuring the protection of individuals' privacy. Methods such as data anonymization and synthetic data generation have been proposed to address this challenge. By leveraging synthetic data, researchers and practitioners can create artificial datasets that safeguard sensitive information while preserving the utility of the data.
  The need for protecting personal data has become increasingly important in today's data-driven world, with privacy concerns [7]–[9] being at the forefront of many discussions. However, the requirement to publish personal data for research and other critical applications has made it essential to find ways to anonymize and protect such data. Synthetic data generation is a promising technique for achieving this goal, where artificial data is generated to replace sensitive data while preserving its statistical properties.
- AutoAugment:
  Automatic augmentation methods have revolutionized the field of computer vision, enabling deep learning models to achieve state-of-the-art performance on a range of challenging tasks. The success of these techniques lies in their ability to augment training data with various transformations and generate diverse and representative samples to improve model generalization.
  The AutoAugment method, introduced by Cubuk et al. [10], has demonstrated the power of automatic search for data-augmentation policies by optimizing a large search space of sub-policies to achieve better accuracy on the

ImageNet and CIFAR-10 datasets. Several studies [11]–[13] have since explored the use of automatic search algorithms to discover optimal data augmentation policies for various vision tasks, demonstrating significant improvements in model performance.

In this paper, we propose a new data set called *** that focuses on caged birds to test the efficacy of automatic data augmentation methods on this domain. Our data augmentation method involved generating transparent cages around birds, which served as a form of regularization during training and helped the model learn the underlying features of birds in cages. We show that our data augmentation method leads to improved performance on a range of computer vision tasks, including image classification and object detection.

We provide an overview of our methodology in Section Purpose Method, where we describe how we generated our caged birds data set and applied data augmentation to enhance model performance. In Section Results on Adding Cage, we present our experimental results and compare the performance of our model with and without our proposed data augmentation technique. Finally, we conclude our paper in Section Conclusion, where we summarize our findings and discuss the potential impact of our approach on the field of computer vision.

III. Proposed Method

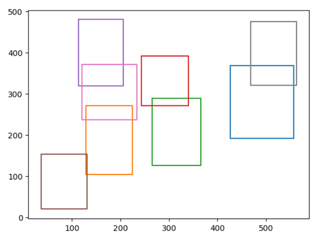
In the first step, we produced cages that are suitable for our method. Initially, we created simple artificial cages using Photoshop tools. This was done to determine if our network could learn from basic types of cages. If the network did not perform well with these cages, then we concluded that it only learned to identify the bird (similar to placing a bird label on the original image).

Although this initial training may seem acceptable, our goal was for the model to learn the general effect of the cage on the bird. Specifically, we wanted the network to understand how the cage bars affect the main photo of the bird.

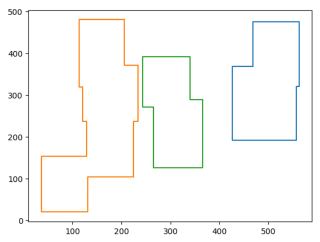
In the next step, we flattened our original dataset to create our synthetic dataset. We used the COCO data-set [14] for our training data, but some of the images in this dataset, such as a cooked chicken or cartoon bird, were not suitable. Therefore, we filtered out these images before adding the cage images to the COCO dataset.

Before adding the cage images to the dataset, we performed a preprocessor on the labeled birds. Each bird had a box label that could overlap with another label. To address this issue, we merged the labels that overlapped with each other. The only computational aspect of this algorithm is related to this part. Based on our reviews, the performance speed of this freehead ensures that the overall training speed will not decrease in any epochs.

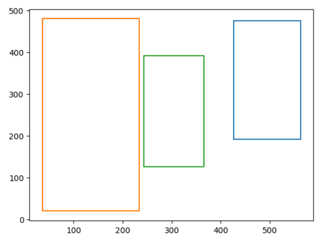
Let's assume that bird labels look like Figure 3a. We reshaped all labels that overlapped with each other because our approach for adding cages required clear labels without any conflicts. Without this step, we might have had overlapping cages after adding the cage, which would have reduced our accuracy.



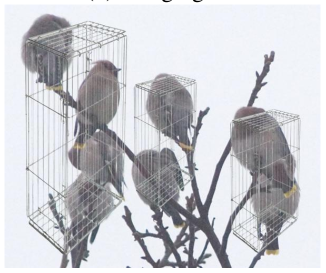
(a) Old labels



(b) Merging labels



(c) New lebels



(d) On Image

Fig. 1: The process of changing the shape of the primary dataset labels

In Figure 3b, we demonstrate our approach for merging labels. At this stage, the labels became polygons and we needed to identify a rectangle that could cover the polygons with overlapping regions, while minimizing the total area. The final result is shown in Figure 1d, where we have successfully reformatted all the labels and added cages to the birds.

To further enhance our performance, we developed three types of hyper-parameters for synthetic data generation, which we used for augmentation:

- **Change Factor**: We observed that even after reshaping the labels, certain parts of the bird were not entirely contained within the cage. This issue could arise due to a variety of factors, such as the labels not fitting the bird, the corners of the cage not being aligned with the edges of the picture, or the cage being curved. By adjusting the scale of the cages based on the Change Factor parameter, we were able to address this problem. Notably, this hyper-parameter was particularly effective in accounting for the variation in bird sizes within the COCO dataset, which includes birds that are close to the camera and occupy a large portion of the image, as well as birds that are far away and appear relatively small.

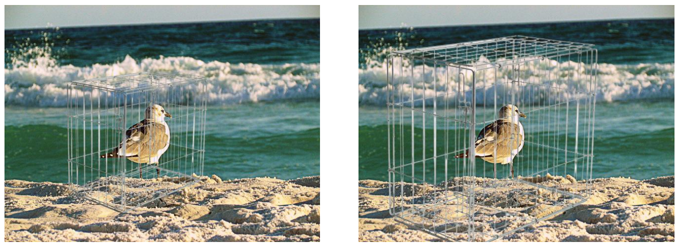

(a) 2x scale (b) 3x scale

Fig. 2: Change Factor Effects

- **Blurriness**: In many real-world scenarios, test images can be blurry due to various reasons such as movement of the subject, camera shake, or motion blur. For instance, in our case study of birds, it is natural to consider that they may move around in their cage during recording, resulting in blurry frames. To address this issue, we used Gaussian blur with a kernel size of 3x3 as part of our augmentation technique. We experimented with three different types of blurring:
  - Blur = 0: No blur was applied to any of the training data.
  - Blur = 1 : We applied blur to all the cages in the training set.
  - Blur = 0 or 1: We randomly applied blur to some cages in each epoch during training.
- **Rotation**: Another important consideration in our approach was the orientation of the cages with respect to the birds. We took this into account by adding a rotation parameter to our labels. This parameter was controlled by two sub-parameters: min-rotate and max-rotate. At the beginning of each epoch, our code selected a random angle between these two values and used it to rotate the cage. We kept the two sub-parameters equal in size and opposite in sign to ensure that the cages were rotated in both clockwise and counterclockwise directions.
- **Percentage of caging**: The percentage of caging is the final hyper-parameter that we utilized in this advanced technique to enhance the robustness of our model for detecting birds inside cages. This specific parameter, which ranges from 0 to 100, was utilized in every epoch when we loaded images for training. We took into account this parameter to determine whether we wanted to apply caging to a particular image or not. Lower values of this parameter resulted in a lower amount of caging being applied. By manipulating this parameter, we were able to effectively control the degree of caging applied to our training images, which in turn enabled us to optimize our model's ability to detect birds inside cages with greater accuracy and precision.

In order to optimize the performance of our model, we employed two distinct methods to identify the most appropriate hyperparameters: the **Greedy** algorithm and the **Genetic** algorithm.

Initially, for the Greedy approach, we considered Blurriness, Rotation (0 - 5x - 10x - 15x), and Change Factor as our target values. Due to the limited range of values in this approach, we were able to exhaustively check all of them, and ultimately found that the Greedy search outperformed other methods such as gene-based systems.

However, as we shifted our focus to include the percentage of caging parameter in our optimization process, we quickly realized that a different approach was required. To optimize this parameter effectively, it was necessary to consider a float value between 0 and 100, making the Genetic algorithm a more appropriate solution. In this new approach, we evaluated all possible values for Rotation, which was only limited to four values (0 - 5x - 10x - 15x) in the Greedy approach.

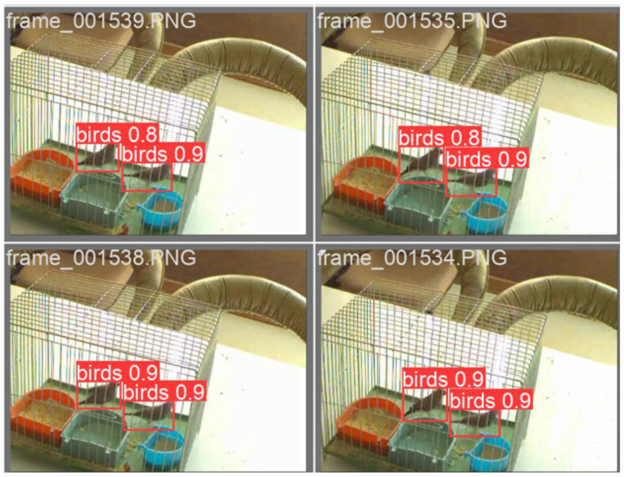
Overall, by using both the Greedy and Genetic algorithms, we were able to more effectively optimize our model's performance, taking into account a broader range of hyperparameters and increasing our chances of achieving the best possible results.

IV. Results on Adding Cage

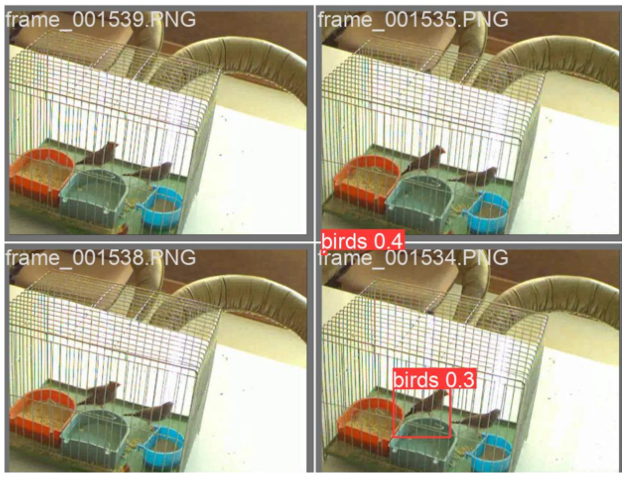
To evaluate the performance of our method, we used Yolov5 models [15], which offer a range of models with varying speeds, including Nano, Small, Medium, Large, and X. To optimize our hyperparameters, we first needed to determine the best format for our cages, which depended on the test format to find out which cages were most suitable for our training purposes. We exhaustively tested all permutations of cages and then considered the effect of our hyperparameters. However, training these models with Large or X models was time-consuming, so we trained them initially with Nano models. We then selected the best-performing models and re-trained them using Small models, and repeated this process until we reached the X model.

To train our models, we used Yolov5 pre-trained models as our fundamental weights and the RTX 2080 Ti for training. However, training our models was challenging, and we did not always achieve the desired results due to the impact of hyperparameters. To overcome this challenge, we trained each of our models four times, which helped us achieve better results and gain a better understanding of our model's performance.

As we discussed in the Purpose and Method section, we also investigated the impact of cage shape and placement on our model's performance. Initially, we used two types of simple cages that we created using Photoshop. However, when we trained our model on synthetic data sets using these cages, we achieved less than 4% accuracy, indicating that simply placing a box or simple cage on the birds would not result in a good model performance on test data. To address this issue, we decided to use real transparent cages, which would provide more accurate representations of the test data cages. Using cages that are similar to the test data cages for data augmentation is an important consideration when training the model.

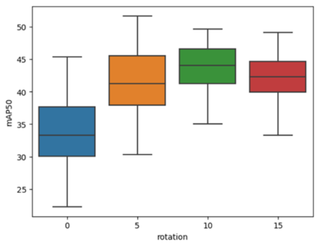


(a) Training with this particular augmentation

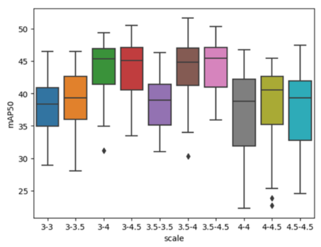


(b) Training with normal form

Fig. 3: Performance difference on test data when using this particular augmentation and not using it



(a) Rotation

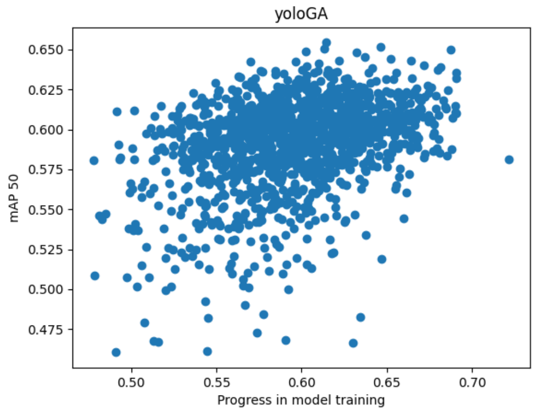


(b) Scale

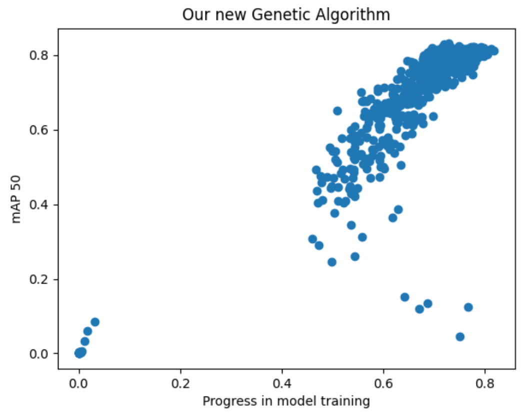
Fig. 4: Nano model result

As discussed in our proposed method, we explored two approaches for identifying the best hyperparameters to use during the training of our model. In the first method, we adopted a greedy approach where we evaluated the impact of rotation and scale parameters on the performance of the model. The results of this evaluation are presented in Figure 4. Our analysis revealed that the optimal range for the scale parameter was between 3x and 4.5x, while the ideal range for the angle parameter was between -15 to 15. By applying these hyperparameters, we continued our training using more robust models, and we were able to achieve an impressive mAP50 score of 60%. This result is a significant improvement of nearly 40% when compared to the base mAP50 score before applying these hyperparameters.

In the second method, we further investigate the use of a genetic algorithm to find optimized values for our model. This time, we include all of the augmentation parameters used in the YOLO model, such as hsv_h, translate, and reset, to ensure a more efficient optimization process. Previous genetic algorithms used in YOLO's base code had poor performance, as they relied solely on mutation algorithms and behaved more like greedy algorithms for tuning hyperparameters. To improve this, we redesigned the evolution process and created a new genetic algorithm code that incorporates crossover, tournament selection, and elite children. Furthermore, we implemented an adaptive method for changing the crossover and mutation rates during the evolution search. To test the effectiveness of our new genetic algorithm, we applied it to the coco128 dataset for tuning the augmentation hyperparameters and compared its results with YOLO's original evolution code. with using the new GA algorithm, after we trained 235 models, we achieved an mAP50 score of over 70% in the 85th model and over 80% in the 235th model and the best model we achieved had an mAP50 score of 83.107%. In contrast, with YOLO GA, we only achieved an mAP50 score of 60% in the 120th model and 60% in the best model.



(a) YOLO's original evolution



(b) New evolution

Fig. 5: Gentic algorithn in yolo model

In Figure 5, we can observe that there is no clear improvement in the mAP values over the course of the training process, which is not ideal. We would typically expect to see a gradual increase in the mAP values as the training progresses. In contrast, the image for the new GA code shows a clear improvement in the mAP values as the training progresses, eventually becoming locally optimized at a certain point. In the yoloGA image, towards the end of the training process, the progress in the results was not visible, and the mAP values appeared to fluctuate randomly.

After achieving these results, we applied the new GA to our main goal of finding appropriate hyperparameters, including the primary hyperparameter for applying the cage technique. Using the new GA, we achieved an mAP50 score of 80%, which is significantly higher than the score obtained using the greedy algorithm. Additionally, we were able to achieve a 20% improvement in the mAP50 score compared to the greedy approach and a 40% improvement compared to not using the cage technique at all.

This investigation clearly demonstrates the superior performance of our GA algorithm over YOLO GA. We are thrilled with this result as it indicates that our approach to hyperparameter tuning was successful and led to a significant improvement in the model's performance. Moreover, we conducted a thorough analysis of the impact of these hyperparameters on the model's performance. Specifically, we examined the effect of varying the hyperparameters on the model's precision and recall scores. Our findings revealed that the optimal hyperparameters not only led to an improvement in the mAP50 score but also increased the precision and recall scores. Our study highlights the importance of hyperparameter tuning in achieving optimal results in machine learning tasks. It demonstrates that a well-designed approach to hyperparameter tuning, such as the one we adopted, can lead to significant improvements in a model's performance.

V. Conclusion

In conclusion, our approach of using augmented cages for training a bird detection model showed promising results in improving the model's performance. Our technique is simple and can be applied to various detection and classification problems where the training data-set is not sufficient. The advantage of our approach is that we add a new feature to the basic data-set that was not present before and we achieve better results without the need to create a new data-set from scratch. Moreover, our approach differs from previous studies in which the generated artificial data were completely separate from the original data. Instead, we showed that by adding an artificial element to the original data, we can improve the performance of the model.The proposed technique can also be extended to other domains. For instance, it can be used for face recognition with masks by combining a dataset of human faces with different types of masks to create an artificial dataset.

In conclusion, our technique has the potential to save time and resources in creating a new dataset from scratch, and it can lead to significant improvements in the performance of the model.

References

[1] Agnieszka Mikołajczyk, Michał Grochowski, Data augmentation for improving deep learning in image classification problem, https://ieeexplore.ieee.org/abstract/document/8388338
[2] Connor Shorten, Taghi M. Khoshgoftaar , A survey on Image Data Augmentation for Deep Learning, https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0/
[3] Jia Shijie, Wang Ping, Jia Peiyi, Hu Siping, Research on data augmentation for image classification based on convolution neural networks,https://ieeexplore.ieee.org/abstract/document/8243510
[4] Jakub Nalepa, Michal Marcinkiewicz, Michal Kawulok, Data Augmentation for Brain-Tumor Segmentation, https://www.frontiersin.org/articles/10.3389/fncom.2019.00083/full
[5] Luis Perez, Jason Wang,The Effectiveness of Data Augmentation in Image Classification using Deep Learning https://arxiv.org/abs/1712.04621
[6] A deep learning approach to detecting volcano deformation from satellite imagery using synthetic datasets, N.Anantrasiri, J.Biggs, F.Albino, D.Bull, https://www.sciencedirect.com/science/article/abs/pii/S003442571930183X
[7] Privacy Preserving Synthetic Data Release Using Deep Learning, Nazmiye Ceren Abay, Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, Latanya Sweeney , https://link.springer.com/chapter/10.1007/978-3-030-10925-7_31
[8] On the Utility of Synthetic Data: An Empirical Evaluation on Machine Learning Tasks, Markus Hittmeir, Andreas Ekelhart, Rudolf Mayer, https://dl.acm.org/doi/abs/10.1145/3339252.3339281
[9] Generating Artificial Data for Private Deep Learning , Aleksei Triastcyn, Boi Faltings, https://arxiv.org/abs/1803.03148

[10] AutoAugment: Learning Augmentation Policies from Data, Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le , https://arxiv.org/abs/1805.09501
[11] Improving Auto-Augment via Augmentation-Wise Weight Sharing, Keyu Tian, Chen Lin, Ming Sun, Luping Zhou, Junjie Yan, Wanli Ouyang, https://proceedings.neurips.cc/paper/2020/hash/dc49dfebb0b00fd44aeff5c60cc1f825-Abstract.html
[12] TrivialAugment: Tuning-Free Yet State-of-the-Art Data Augmentation ,Samuel G. Müller, Frank Hutter, https://openaccess.thecvf.com/content/ICCV2021/html/Muller_TrivialAugment_Tuning-Free_Yet_State-of-the-Art_Data_Augmentation_ICCV_2021_paper.html
[13] Online Hyper-Parameter Learning for Auto-Augmentation Strategy, Chen Lin, Minghao Guo, Chuming Li, Xin Yuan, Wei Wu, Junjie Yan, Dahua Lin, Wanli Ouyang, https://openaccess.thecvf.com/content_ICCV_2019/html/Lin_Online_Hyper-Parameter_Learning_for_Auto-Augmentation_Strategy_ICCV_2019_paper.html.
[14] COCO data set, https://cocodataset.org/home.
[15] Glenn Jocher ,"YOLOv5", https://github.com/ultralytics/yolov5.