Data Augmentation Techniques for Airbus Ship Detection: A Comparative Study on the Effects of Different Methods on the Accuracy and Robustness of the Trained Models

Simone Carena s309521@studenti.polito.it
s309521

Francesco Paolo Carmone s308126@studenti.polito.it
s308126

Ludovica Mazzucco s315093@studenti.polito.it
s315093

Abstract

Data augmentation plays a pivotal role in enhancing the performance and robustness of any Machine Learning model. By artificially expanding the size and diversity of the training dataset, data augmentation mitigates overfitting and equips the model with a broader understanding of variations within the input data. This article describes several techniques for data augmentation applied to an Object Detection task and their effect on the robustness and accuracy of the resulting model.

Keywords: Data Augmentation, Object Detection, Robustness, Accuracy

1. Introduction

Object Detection plays a major role in many real-world scenarios, ranging from autonomous driving to modern agriculture, the applications of this computer vision field are countless. A common approach to this challenge lies in the use convolutional neural network, capable of discovering patterns in the images and deriving the requested information. Although this technique is widely used and generally successful on i.i.d. data, these models' performance may degrade following distributional shifts in the input images, caused by common corruptions that may occur in a real-world setting. These naturally occurring distortions, like the presence of fog or motion blur in the captured pictures, could undermine the efficacy of a trained model, even though it might perform well on the original dataset, thus limiting its applicability in real scenarios.

Given these premises, our work focuses on the Kaggle's Airbus Ship Detection challenge, with the aim of expanding its scope outside the base object detection task, studying the effects common corruptions on the images have on the network and how different data augmentation methods may improve its robustness. In particular we show what may be the underlying reasons why some techniques generate more robust models, and how the robustness of such models may not always be conciliable with the accuracy on the clean dataset.

The code implementation is made available on GitHub.

1.1. The Robustness vs Accuracy Trade-off

Possible approaches to solve the issue of generating robust models would be to introduce

architectural changes to the network, or to carry out data augmentation techniques to allow the model to learn domain-invariant features, allowing it to be more robust to distributional shifts. Though the data augmentation methods may lead, in general, to more robust models, this could come at the cost of an accuracy loss on the original data. As the authors in Tsipras et al. (2018) show, there may exist an inherent trade-off between accuracy and robustness for a network, explained in terms of the different features representation learnt by a standard model.

To further analyze the effect of image corruptions on the performance of the models, the authors in Yin et al. (2019) perform a Fourier analysis on the perturbations that may affect the images, highlighting how some corruptions affect the lower end of the spectrum of the images, while others affect higher frequencies. This suggests that training a model using augmentations that affect certain areas of the spectrum may lead to increased robustness when faced with distortions that fall in the same area of the spectrum. They also show how some type of augmentations can render the model more robust to certain corruptions, while rendering it less effective against other types of disturbances. What they suggest to this end is in fact to apply more varied types of data augmentations, in order to try covering wider areas of the spectrum.

To address these problems, in the following chapters are presented techniques of data augmentation that try to render the models more robust against distributional shifts, while also preserving the accuracy on the standard images.

2. Data

The dataset utilized is from the Kaggle challenge Airbus Ship Detection, originally designed for a Semantic Segmentation Task. It includes over 208,000 images in `.jpeg` format, totaling approximately 31GB of data. The file `train_ship_segmentations.csv` provides the ground truth in run-length encoding format for 192,000 of these images, leaving the remainder unlabeled. For our purposes, we sampled 40% of the dataset for training, resulting in around 77,000 images. Both the validation and test sets comprised 13.33% of the total dataset, roughly 26,000 images each. Importantly, the training, validation, and test sets are mutually exclusive, containing no overlapping images. Due to constraints on training time, we were limited to using only this subset of the data. Training the model for 18 epochs with this amount of data took approximately 11 hours, which is close to Kaggle's 12-hour runtime limitation. The network implementation includes automatic normalization, which requires calculating the mean and standard deviation of the dataset. This involves computing these statistics for each channel across the entire dataset. Kaggle also imposes a RAM consumption limit of 30GB, influencing the batch size. Larger batch sizes increase RAM usage due to more images being loaded simultaneously. After several trials, we settled on a batch size of 32.

2.1. Ground truths

The ground truth data initially provided for the instance segmentation task consisted of contour lines delineating the boundaries of each mask. To adapt this data to the *Faster R-CNN* framework, which requires bounding box annotations in a specific format, extensive pre-processing was undertaken. The created script facilitated the conversion of the input data from its original `.csv` format to the required `.npy` format, ensuring compatibility with the *Faster R-CNN* model. The conversion to the `.npy` format wasn't strictly necessary, as the data could have been trans-

lated on-demand during runtime. However, opting for this approach would have introduced unnecessary overhead. Instead, by pre-processing the data and producing the `.npy` file once, which took approximately 10 minutes, we were able to significantly reduce loading time during subsequent model runs. This optimization allowed us to streamline the training process without sacrificing runtime efficiency.

2.2. Data Augmentation

Auxiliary Fourier Basis Augmentation

The first proposed data augmentation comes from Vaish et al. (2024), where the authors note that even if trained with visual augmentations, models can still be sensitive to image variations not included in the training. In accordance to Yin et al. (2019), they attribute this behavior to the frequency components of the augmentations performed, which do not necessarily cover the whole spectrum of possible image distortions. In order to overcome this issue they suggest to complement the visual augmentation strategies with explicit use of Fourier basis functions (5). The basis functions added to the image, one per individual channel, are sampled as following: first a frequency f and an amplitude ω are sampled from a uniform distribution as $f \sim \mathcal{U}_{[1,M]}$ (where M is the size of the image) and $\omega \sim \mathcal{U}_{[0,\pi]}$. The resulting basis functions are then added to the original image, each re-scaled by a weight factor σ, sampled from an exponential distribution $\sigma \sim \text{Exp}\left(\lambda^{-1}\right)$. The choice of the exponential distribution is motivated by the facts that it renders perturbations with larger magnitude less and less likely. The resulting image will be

$$X_c = \text{clamp}_{[0,1]}\left(X_c + \sigma_c A_{w_c,f_c}\right), \quad c \in [\text{R}, \text{G}, \text{B}] \tag{1}$$

Patch Gaussian Augmentation

The authors in Lopes et al. (2019), underlying the trade-off the arises between robustness and accuracy when trying to train robust models, propose a new augmentation scheme, *Patch Gaussian*, that adds noise to randomly selected patches in an input image. The focus of this paper, and the related data augmentation technique, is to render the models not only accurate on the original image distribution, but also robust to distributional shifts caused by various forms of image corruptions.

The method select a random $W \times W$ patch within the image bound, generates a Gaussian noise of variance σ_n^2, and applies it to the original image. The coordinates of the center of the image are chosen uniformly such that the patch is all inside the image bounds, as $(c_x, c_y) \sim \mathcal{U}_{[0,w-W]\times[0,h-W]}$, where w and h are the dimensions of the image. Let $\mathbf{X} \in [0,1]^{w,h,3}$ be the original image, and $\mathbf{G} \in [0,1]^{w,h,3}$ a tensor composed of all 0 except in $[c_x - W/2, c_x + W/2] \times [c_y - W/2, c_y + W/2]$ (the area is the same for each layer), where each pixel p is sampled from a Normal distribution as $p \sim \mathcal{N}(0, \sigma_n^2)$, then the resulting augmented image $\tilde{\mathbf{X}}$ is defined

$$\tilde{\mathbf{X}} = \text{clamp}_{[0,1]}\left(\mathbf{X} + \mathbf{G}\right) \tag{2}$$

Geometric Transformations

The last data augmentation method consisted in only using geometric transformations of the images, with the care of not using too aggressive ones, as we saw they could make the ships disappear in certain cases. After some trials we chose to use the following: random perspective, padding, random horizontal and random vertical flip. We had to exclude random rotations and random affine transformations, as often we noticed the ships would fall outside the frame.

3. Methods

This section is focused on the description of the approach used in order to fulfill the goal described in previous sections.

3.1. Model choice

The first challenge we encountered was deciding whether to create a network from scratch or to use an existing one for transfer learning. We preferred the latter option, as it seemed more efficient to focus on optimizing a pre-trained model rather than spending time and computational resources on generating a new one, which might have incorrect settings. Furthermore, the limited resources available to us, such as the Kaggle GPU P100, did not allow for extended training periods. Consequently, training a robust model under these conditions was not feasible.

Hence, after a careful review of the principal machine learning algorithms for vision tasks studied during the course, we initially chose YOLO. However, we noticed that it was already highly optimized, leaving little room for improvement. When we tested it with our training images, it was already able to detect the ships satisfactorily. In order to not trivialize our work, we preferred the Faster-RCNN [1], a per se efficient algorithm for object detection whose base structure is provided by `PyTorch`, a fundamental feature for us since this allowed us to dive into the code and understand better the backend functionalities and how to customize it for our purposes.

1. See here https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html#torchvision.models.detection.fasterrcnn_resnet50_fpn

3.2. Network Architecture

As the previous paragraph has anticipated, we adopt the Faster R-CNN as our base model, enhanced with the inclusion of batch normalization layers. The *Faster R-CNN* is a network built by integrating three key components: a *ResNet* backbone for feature extraction, the *Region Proposal Network (RPN)* that looks for the most probable regions containing bounding boxes, and the *Region of Interest (ROI)* heads that contain the box predictor, as shown in 1. The *RPN* efficiently generates region proposals to focus on potential object regions. This eliminates the need for external algorithms like Selective Search, reducing computational overhead. A deep CNN then locates the ROIs inside the regions output by the previous module, passes them to a pooling layer, in which all information contained in the region pixels is squeezed to extract meaningful features, and lastly a box predictor head elaborates this data through a dense layer to finally predict the target objects bounding boxes with the corresponding labels. This box predictor is indeed the component we changed in order to have only two output labels, 0 for `background` and 1 for `ship`. By unifying these components into a cohesive end-to-end architecture, the *Faster R-CNN* not only realizes notable enhancements in training loss but also stands as a satisfactory solution for addressing our specific needs.

3.3. Batch normalization

In the initial stages of training, we encountered persistent stagnation in training losses, irrespective of the number of epochs trained. To address this issue, we adopted a strategy inspired by the findings presented in Zhao et al. (2017). Specifically, we incorporated batch normalization layers into each convolutional network. This adjustment aimed to stabilize the learning process by normaliz-

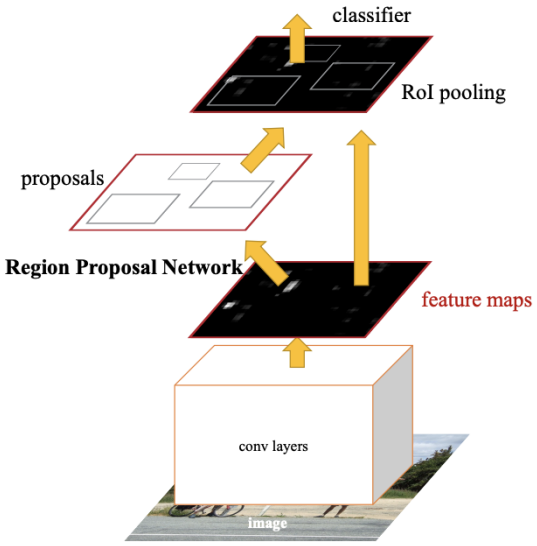


Figure 1: Faster-RCNN Network Architecture

ing the activations within each mini-batch, thereby enhancing model convergence and performance.

3.4. Preliminaries

Fourier Transform

Let $X \in [0,1]^{w,h}$ be a matrix, representing one of the channels of the RBG image we are considering, where we assume the pixels can assume values in the range $[0,1]$. Let $\mathcal{F} : \mathbb{R}^{w,h} \to \mathbb{C}^{w,h}$ the *Discrete Fourier Transform* (DFT) of our matrix X, and $\mathcal{F}^{-1}$ its inverse. The transformed matrix $F = \mathcal{F}(X)$, can be computed as

$$F[u,v] = \frac{1}{wh} \sum_{n=0}^{w-1} \sum_{m=0}^{h-1} X[m,n] e^{-j2\pi\left(\frac{u}{w}n + \frac{v}{h}m\right)} \quad (3)$$

Each element $F[u,v] \in \mathbb{C}$ can be represented in its polar form as $F[u,v] = r \cdot \exp(j\theta)$, where $r = ||F[u,v]||$ and $\theta = \arctan(\Re(F[u,v])/\Im(F[u,v]))$. The computation of the Fourier Transform of the images is performed via the Fast Fourier Transform (FFT) algorithm, available in `pytorch` in the `fft` module as `fft2` and `ifft2` for the transform and its inverse respectively. The Fourier transform of an RGB image consists of the transform of each individual channel: let $\mathbf{X} \in [0,1]^{w,h,3} = (X_R, X_G, X_B)$, then its DFT $\mathbf{F} = \mathcal{F}(\mathbf{X})$ can be computes as

$$\mathbf{F} = \mathcal{F}(\mathbf{X}) = (\mathcal{F}(X_R), \mathcal{F}(X_G), \mathcal{F}(X_B)) \quad (4)$$

Fourier Basis Functions

A *Fourier Basis Function*, as described in Vaish et al. (2024), is a planar wave whose amplitude $A(u,v)$ at position (u,v) can be expressed as

$$A_{f,\omega}(u,v) = R \sin\left(2\pi f \left(u\cos(\omega) + v\sin(\omega) - \frac{\pi}{4}\right)\right) \quad (5)$$

where the parameters f and ω represent, respectively, the frequency and the direction of the resulting wave. A Fourier basis function characterized as above, can be expressed in the frequency domain as a delta function.

3.5. Hyper Parameters and Loss Function

The following parameters were chosen via a trial-and-error methodology. A batch size of 32, due to Kaggle constraints, as deeply explained in 2 A varying Learning Rate scheduled according to a step function, that starts from `1e-4` and decreases at each epoch. The loss function used by the network is a sum of four terms: a Cross Entropy, a Smooth ℓ_1, which is used twice in the computations, and Binary Cross-Entropy with logits.

Training Transforms

In what regards hyper-parameters used for the training transforms, the Patch Gaussian

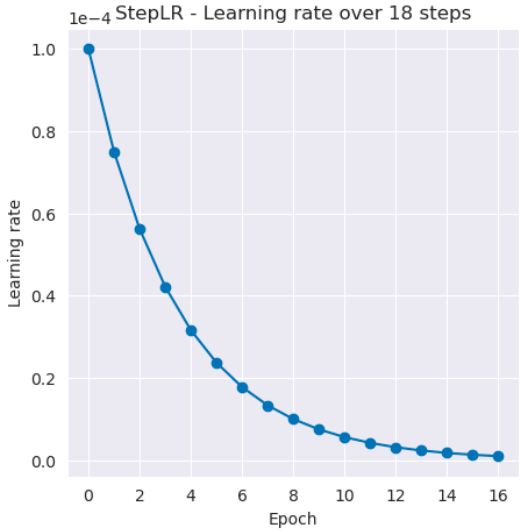


Figure 2: Learning Rate Scheduler

uses a patch size of 30 and a σ^2 of 0.2, while the Fourier Basis Augmentation samples the wave amplitude from an exponential distribution with rate $\lambda = 0.3$.

Test Transforms

for the rest part of transforms, the ones exploited during test phase, the following hyper-parameters have been selected:

- **Gaussian Noise**: $\sigma^2 = 0.1$;
- **Gaussian Blur**: using `GaussianBlur` from `torchvision.transforms`, with kernel size = 7 and standard deviation $\sigma = 5$;
- **Fog** : with density of the fog = 0.5 [2];
- **Motion Blur**: kernel size = 16 and a direction angle = 45°;
- **Brightness**: exploiting `adjust_brightness` from `torchvision.transforms.functional` with a brightness factor = 1.5.

3.6. Network Training

The *Faster R-CNN* model expects input in the form of a list of dictionaries, where each dictionary contains keys for `boxes` and `labels`, with corresponding tensor values. The `boxes` tensor comprises four values ranging from 0 to 224 (i.e, our network input images dimension), representing the coordinates of the top-left and bottom-right corners of the bounding box, scaled accordingly to the network input images.

The network was trained for 18 epochs on 40% of the entire dataset. The model was first trained by freezing the backbone layers, and training the box predictor; then a fine-tuning of the entire network, with a lower learning rate of `1e-6`, was performed.

We trained in total four different models, each one using the different data augmentations presented throughout the paper.

3.7. Evaluation Strategies

To evaluate the model's performance we used an independent test set instead of a k-fold cross-validation strategy, as it would have affected too much the training time, which was already problematic due to Kaggle's constraints. Even though we did not use the more sophisticated k-fold technique, the independent test set has proven to be efficient enough for our purposes and we did not encounter over-fitting nor any generalization issues. This happened especially because of the great size of the dataset, which allowed us to assume that the test set distribution was not skewed. In order to evaluate the resulting model, we use the Mean Average Precision (mAP) metric, as it was deemed

2. Instead of generating the fog exploiting an additional network to estimate the field depth, we simply added a layer of uniform white fog over the image.

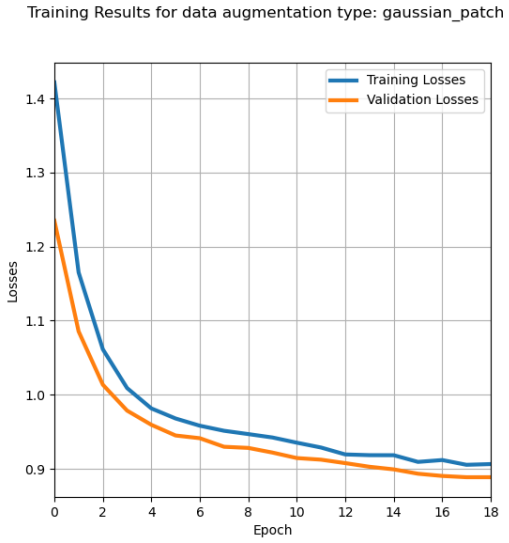


Figure 3: Training losses in the Patch Gaussian models. The others, performed similarly.

the most suitable one, and most widely used in Object Detection tasks.

3.8. Challenges encountered

Regarding the main difficulties we faced, we distinctly remember the initial challenges. Firstly, we had to convert the original instance segmentation labels from a running-length format to bounding boxes formatted as 'XYXY', which is widely used by object detection functions for better readability. Secondly, the training set lacked clear indexing for samples, causing mismatches between inputs and targets. This issue necessitated the use of a sequential data loader without shuffling to resolve. Lastly, we encountered issues with transformations applied to bounding boxes, particularly geometric transformations, which seemed to affect only images while leaving bounding boxes in their original positions. It became evident that the solution lay in treating bounding boxes as instances of the BoundingBoxes class rather than standard Tensors.

4. Experiments

Following the benchmark guidelines to test the robustness of a model provided in Michaelis et al. (2019), 5 image distortion techniques were implemented to simulate scenarios that may occur in a real-world setting. The presented corruptions, as shown in 4, are: brightness modifications, to simulate changes in the lightning of the environment; fog covering the image, to make up for partial occlusions to the scene due to fog or similar atmospheric conditions; Gaussian blur and motion blur, to recreate out-of-focus lenses or motion distortions; Gaussian noise, to account for possible damages/imperfections in the retrieved pictures.

Each of the trained model was tested both on the "clean" original dataset, and on every corrupted version of it, in order to asses its robustness against naturally occurring distortions and its accuracy on the original images distribution. The results of the experiments can be seen in 1.

As shown in 5, the data augmentation techniques presented in Vaish et al. (2024) and Lopes et al. (2019), affect different areas of the Fourier spectrum[3] of the images. At the same time, the test images also present a distorted spectrum with respect to the original image as shown in 5, in accordance to Yin et al. (2019).

5. Conclusions

Table 1 displays the Mean Average Precision (mAP) results at an IoU threshold of 50%, following Non-Maximum Suppression (NMS)

3. The Fourier spectrum was derived using 3 as described in 4. The presented images are the spectrum of the grey-scale of the transformed image.

on bounding box outputs. The model trained using Fourier augmentation on the training set demonstrates superior performance across various image corruptions, except for scenarios labeled *Original* and *Brightness*. In these cases, the model trained with Patch Gaussian augmentation achieves the highest performance. This discrepancy arises from differences in neuron activation patterns during training; Fourier basis transforms may not effectively capture changes in brightness compared to the baseline model. As discussed earlier, these results are consistent with research on Fourier analysis, indicating that different corruptions impact distinct spectral regions, thereby making the network more susceptible to distortions in untrained areas. Fourier augmentation potentially enhances performance by highlighting a broader range of spectral areas, as detailed in the original paper. While this study demonstrates that integrating Fourier and Patch Gaussian augmentations during model training improves robustness against real-world image distortions compared to standard techniques, achieving high mAP values (peaking around 0.5) was challenging due to limited access to Kaggle GPUs. Longer training and more extensive data could further enhance model performance across various image distortions.

References

Raphael Gontijo Lopes, Dong Yin, Ben Poole, Justin Gilmer, and Ekin D. Cubuk. Improving robustness without sacrificing accuracy with patch gaussian augmentation. *CoRR*, abs/1906.02611, 2019. URL http://arxiv.org/abs/1906.02611.

Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

Puru Vaish, Shunxin Wang, and Nicola Strisciuglio. Fourier-basis functions to bridge augmentation gap: Rethinking frequency augmentation in image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17763–17772, June 2024.

Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin Dogus Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/b05b57f6add810d3b7490866d74c0053\-Paper.pdf.

Zhong-Qiu Zhao, Haiman Bian, Donghui Hu, Wenjuan Cheng, and Hervé Glotin. Pedestrian detection based on fast r-cnn and batch normalization. In *Intelligent Computing Theories and Application: 13th International Conference, ICIC 2017, Liverpool, UK, August 7-10, 2017, Proceedings, Part I 13*, pages 735–746. Springer, 2017.

Figure 4: Image corruptions that may occur in a real world scenario, due to atmospheric conditions or malfunctions in the camera systems.

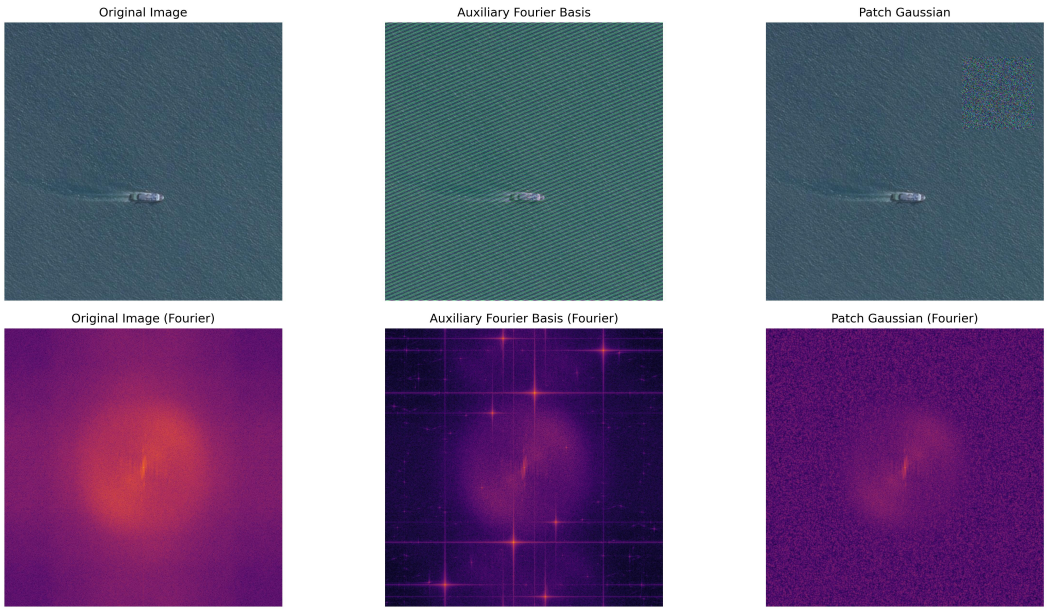


Figure 5: Fourier Spectrum of the Different Data Augmentation Techniques

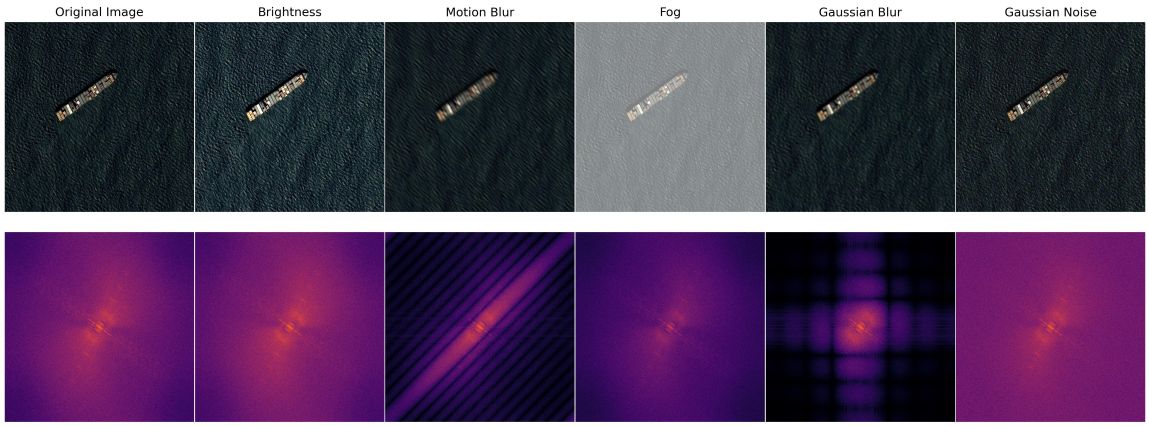


Figure 6: Fourier Spectrum of the Corrupted Test Images

	Original	**Fog**	**Motion Blur**
None	0.4630	0.1152	0.0837
Geometric	0.4280	0.0749	0.0745
Fourier Basis	0.4550	0.0874	0.0772
Patch Gaussian	0.4733	0.0784	0.0791

	Gaussian Blur	**Gaussian Noise**	**Brightness**
None	0.2273	0.2518	0.3500
Geometric	0.2007	0.2250	0.2814
Fourier Basis	0.2573	0.2956	0.2729
Patch Gaussian	0.2441	0.2788	0.3805

Table 1: mAP results of the different trained models

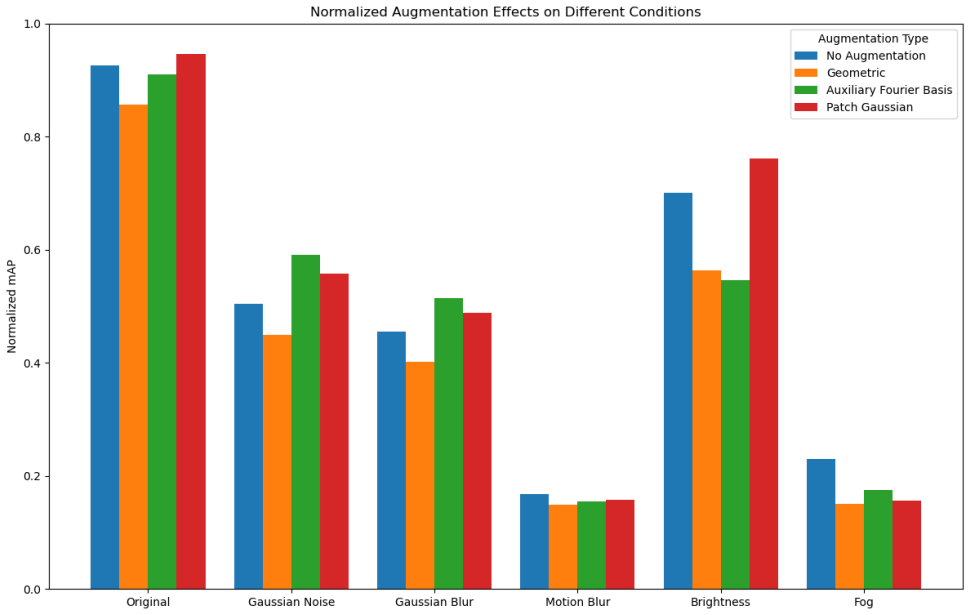
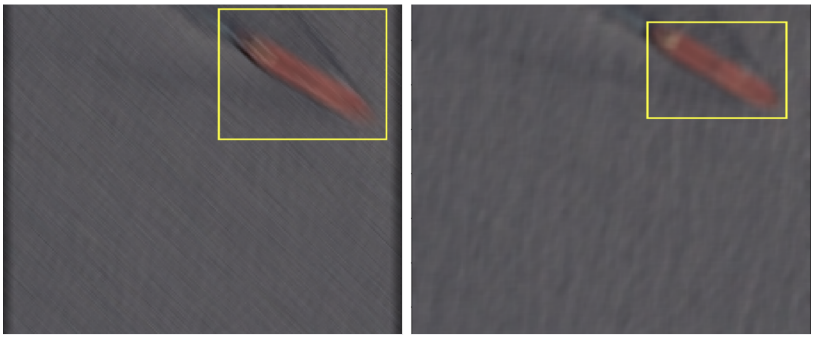
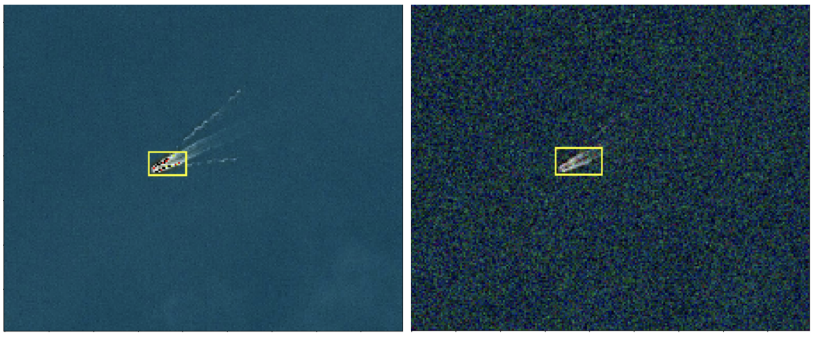


Figure 7: Bar Chart of the test results of the different trained models



(a) Patch Gaussian, Motion Blur

(b) Fourier Basis Augmentation, Gaussian Blur



(c) Patch Gaussian, Brightness

(d) Fourier Basis Augmentation, Gaussian Noise