TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Aleksandr Ivanov 211489IAPM

Overcoming Dataset Limitations: Advanced Augmentation Techniques for Fish Species Classification with Convolutional Neural Networks

Master's Thesis

Supervisor: Elizaveta Dubrovinskaya
PhD

Co-supervisor: Jeffrey Andrew Tuhtan
Associate Professor

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksandr Ivanov 211489IAPM

ANDMEKOGUMITE PIIRANGUTE ÜLETAMINE: ARENENUD SUURENDAMISTEHNIKAD KALALIIKIDE KLASSIFIKATSIOONIKS KONVOLUTSIOONILISTE NEURONAALSETE VÕRKUDE ABIL

Magistritöö

Juhendaja: Elizaveta Dubrovinskaya
PhD

Kaasjuhendaja: Jeffrey Andrew Tuhtan
Associate Professor

Tallinn 2023

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksandr Ivanov

07.05.2023

Abstract

Fish species classification is a critical task in underwater ecosystem monitoring, It remains highly challenging due to the limited availability and poor quality of datasets. Convolutional neural networks (CNNs) have shown promising results in fish species classification, but their performance heavily depends on the quality and size of datasets. This study has investigated the effectiveness of various image augmentation techniques in improving the performance of a specific CNN model for fish species multi-class classification in the context of limited, low-quality datasets. The author compared traditional augmentation techniques with novel ones, such as Stable Diffusion, and evaluated their impact on the classification model's performance and time costs. This study has also explored the feasibility of applying these techniques to small initial datasets and assessed their effectiveness in mitigating issues related to dataset imbalances and overfitting.

Achieved findings have suggested that a combination of synthetic images generated using Stable Diffusion, fine-tuned with *Supplementary* and *AFFiNe* datasets, along with additional *RandAugment* transformations, yielded the best performance improvements in terms of evaluation metrics of *ResNet18* and *ResNet50* models. It was also discovered that advanced image augmentation techniques, such as Stable Diffusion, can be effectively applied in limited data scenarios, contributing to the development of robust fish species classification models.

The results of this study have implications for the field of fish species classification, including the potential for novel augmentation techniques to be worth applying and the contribution of advanced models to monitor and preserve underwater ecosystems. Future work could focus on refining the proposed solution by improving the prompts used, training custom image upscaling models, or investigating alternative augmentation techniques. Ultimately, the study demonstrates the potential for advanced augmentation techniques to make a significant contribution to the development of robust fish species classification models.

The thesis is written in English and is 57 pages long, including 8 chapters, 65 figures, and 20 tables.

Annotatsioon

Andmekogumite piirangute ületamine: Arenenud suurendamistehnikad kalaliikide klassifikatsiooniks konvolutsiooniliste neuronaalsete võrkude abil

Kalaliikide klassifitseerimine on veealuste ökosüsteemide jälgimisel kriitiline ülesanne, kuid piiratud ja madala kvaliteediga andmekogumite tõttu on see keeruline. Konvolutsioonilised neuronaalsed võrgud (CNN-id) näitavad selles valdkonnas paljulubavaid tulemusi, kuid nende jõudlus sõltub andmekogumite kvaliteedist ja suurusest. Käesolev uuring uuris pildi suurendamise tehnikate tõhusust CNN-mudeli jõudluse parandamisel kalaliikide klassifikatsioonil, võrreldes traditsioonilisi tehnikaid uutega, nagu stabiilne difusioon.

Uuring näitas, et stabiilse difusiooni, peenhäälestatud andmekogumite ja täiendavate transformatsioonidega genereeritud sünteetiliste piltide kombinatsioon parandas kõige enam *ResNet18* ja *ResNet50* mudelite jõudlust. Stabiilne difusioon osutus tõhusaks ka piiratud andmetega stsenaariumides, aidates kaasa tugevate kalaliikide klassifikatsioonimudelite arendamisele.

Uuringu tulemused viitavad uute suurendamistehnikate väärtusele kalaliikide klassifikatsiooni valdkonnas ning nende panusele veealuste ökosüsteemide jälgimise ja säilitamise arendamisel. Edasised uurimissuunad võivad keskenduda lahenduse täiustamisele, kohandatud pildi suurendamise mudelite koolitamisele või alternatiivsete tehnikate uurimisele. Lõppkokkuvõttes näitab uuring arenenud suurendamistehnikate potentsiaali panustada oluliselt tugevate kalaliikide klassifikatsioonimudelite arendamisse.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 57 leheküljel, 8 peatükki, 65 joonist, 20 tabelit.

List of Abbreviations and Terms

AA	AutoAugment
ANN	Artificial Neural Network
BfG	The German Federal Institute of Hydrology (Bundesanstalt für Gewässerkunde)
CNN	Convolutional Neural Network
CPU	Computer’s processor
CV	Computer Vision
Frame	A part of an image or an individual image of a video at a specific moment of time
FP	False Positive
FN	False Negative
GAN	Generative Adversarial Network
GPU	Graphics card
Hyperparameter	External model parameter, that cannot be estimated from data, used to control the learning process
RA	RandAugment
RL	Reinforcement Learning
SD	Stable Diffusion
SVD	Singular Value Decomposition
TP	True Positive
TN	True Negative
UI	User Interface
VAE	Variational Autoencoder
VRAM	GPU’s memory
YOLO	You only look once. Object detection model

Table of Contents

1 Introduction **11**
- 1.1 Related literature 12
- 1.2 Problem statement 15
- 1.3 Author contribution and scope 16
- 1.4 Thesis structure 17

2 Preliminaries **18**
- 2.1 CNN model selection 18
  - 2.1.1 ResNet 20
- 2.2 Data augmentation pipeline 21
- 2.3 Basic augmentation 22
- 2.4 Advanced augmentation 22
  - 2.4.1 AutoAugment 23
  - 2.4.2 Fast and Faster AutoAugment 24
  - 2.4.3 RandAugment 25
  - 2.4.4 AugMix 26
- 2.5 Stable Diffusion 27
  - 2.5.1 Stable Diffusion v1.5 and v2.1 28
  - 2.5.2 Fine-tuning: DreamBooth 29
  - 2.5.3 Fine-tuning: LoRA 30
- 2.6 Model training 30
- 2.7 Validation 31

3 Datasets **33**
- 3.1 Supplementary 33
  - 3.1.1 Challenges 34
  - 3.1.2 Preprocessing 35
- 3.2 AFFiNe 37

4 Experiments **39**
- 4.1 Experimental setup 39
- 4.2 Data and model preparation 40
  - 4.2.1 Dataset 40
  - 4.2.2 Data transformation 41
  - 4.2.3 Dataloader and Data sampler 42

4.2.4 Model preparation . . . 43
4.3 BASELINE model training . . . 44
4.4 AutoAugment . . . 44
4.5 Faster AutoAugment . . . 45
4.5.1 Optimal policy searching . . . 45
4.5.2 Model training . . . 45
4.6 RandAugment . . . 46
4.7 AugMix . . . 46
4.8 Stable Diffusion: Data preparation . . . 47
4.8.1 Supplementary . . . 47
4.8.2 AFFiNe . . . 49
4.8.3 Images captioning . . . 49
4.9 Fine-tuning: DreamBooth . . . 51
4.10 Fine-tuning: LoRA . . . 51
4.11 Checkpoint selection . . . 53
4.12 Synthetic datasets generation . . . 54
4.13 Model training using synthetic datasets . . . 54
4.14 Model training using a mix of augmentations . . . 54

5 Validation and results . . . 55
5.1 BASELINE model evaluation . . . 55
5.2 Model training with augmentation evaluation . . . 56
5.2.1 Selection of RandAugment . . . 57
5.2.2 Selection of AugMix . . . 58
5.2.3 Faster AutoAugment search . . . 59
5.2.4 Synthetic datasets . . . 61
5.2.5 Total metrics comparison . . . 62
5.2.6 Total spent time comparison . . . 63
5.3 Best models evaluation . . . 64
5.4 Problem analysis . . . 65

6 Discussion and further work . . . 66

7 Summary . . . 67

8 Acknowledgements . . . 68

References . . . 69

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis . . . 73

Appendix 2 – Annotation tool 74

Appendix 3 – Caption tool 75

Appendix 4 – Custom dataset class 76

Appendix 5 – Custom training loop 77

Appendix 6 – SD Supplementary fine-tuning dataset 79

Appendix 7 – SD AFFiNe fine-tuning dataset 80

Appendix 8 – RunPod post-installation script 81

Appendix 9 – Checkpoint selection 82

Appendix 10 – Synthetic datasets 88

List of Figures

1 Comparison of GAN-based augmentation to latent diffusion [23]. 15

2 ANN structure [24]. 18
3 Principe of convolutional layer[1]. 19
4 Typical CNN structure[2]. 19
5 Comparison of plain CNN and ResNet layers [25]. 20
6 A block with residual connection [25]. 20
7 CNN model training pipeline without augmentation. 21
8 CNN model training pipeline with augmentation. 21
9 Examples of basic augmentation techniques [7]. 22
10 AutoAugment best policy search optimization cycle [28]. 23
11 An procedure of best policies search in Fast AA [30]. 24
12 Differentiable data augmentation training in Faster AA [31]. 25
13 Example images augmented by RandAugment [29]. 26
14 Example of AugMix application on an image [32]. 26
15 Example of AugMix application on an image [33]. 27
16 Example of AugMix application on an image [11]. 28
17 U-Net noise predictor with text attention layers[3]. 28
18 Comparison of "*photo if an eel*" generated by SD and a real photo. 29
19 Fine-tuning Stable Diffusion using DreamBooth. 29
20 Example of Underfitting and Overfitting[4]. 31
21 Example of confusion matrix [39]. 32

22 *Supplementary* dataset raw frames examples [40]. 33
23 Examples of challenges with *Supplementary* dataset [40]. 34
24 *Supplementary* dataset distribution after automatic extraction [40]. 35
25 *Supplementary* dataset frames examples after prepossessing [40]. 36
26 *Supplementary* dataset distribution after prepossessing [40]. 36
27 *AFFiNE* dataset raw frames examples [34]. 37
28 *AFFiNE* dataset distribution [34]. 37

29 Example of test subset images [40]. 40
30 Visual comparison of a default and a new transformation [40]. 42
31 Difference of Undersampling and Oversampling[5]. 42
32 Example of a synthetic *bleak* fish generated from *bad* training samples. . . 48
33 Example of various upscaling models in *stable-diffusion-webui*. 48

34 Baseline *ResNet18* model training and validation accuracy. 55
35 Baseline *ResNet50* model training and validation accuracy. 55
36 Normalized confusion matrices of baseline models. 56
37 Faster AA search training loss. 60
38 Normalized confusion matrices of *SD_SUP_COMBINED_RAND* models. 64
39 Examples of images with an incorrectly predicted label [40]. 65

40 Annotation tool [40]. 74
41 Image description caption tool [40]. 75
42 Supplementary images used for fine-tuning Stable Diffusion [40]. 79
43 AFFiNe images used for fine-tuning Stable Diffusion [34]. 80
44 SD v1.5 *Supplementary* checkpoint analysis (*LoRA*). 82
45 SD v1.5 *Supplementary* checkpoint analysis (*DreamBooth*). 82
46 SD v2.1 *Supplementary* checkpoint analysis (*LoRA*). 83
47 SD v2.1 *Supplementary* checkpoint analysis (*DreamBooth*). 83
48 SD v1.5 *AFFiNe* checkpoint analysis (*LoRA*). 84
49 SD v1.5 *AFFiNe* checkpoint analysis (*DreamBooth*). 84
50 SD v2.1 *AFFiNe* checkpoint analysis (*LoRA*). 85
51 SD v2.1 *AFFiNe* checkpoint analysis (*DreamBooth*). 85
52 SD v1.5 *Supplementary+AFFiNe* checkpoint analysis (*LoRA*). 86
53 SD v1.5 *Supplementary+AFFiNe* checkpoint analysis (*DreamBooth*). . . 86
54 SD v2.1 *Supplementary+AFFiNe* checkpoint analysis (*LoRA*). 87
55 SD v2.1 *Supplementary+AFFiNe* checkpoint analysis (*DreamBooth*). . . 87
56 SD v1.5 *Supplementary* synthetic images (*LoRA*). 88
57 SD v1.5 *Supplementary* synthetic images (*DreamBooth*). 89
58 SD v2.1 *Supplementary* synthetic images (*LoRA*). 90
59 SD v2.1 *Supplementary* synthetic images (*DreamBooth*). 91
60 SD v1.5 *Supplementary+AFFiNe* synthetic images (*LoRA*). 92
61 SD v1.5 *Supplementary+AFFiNe* synthetic images (*DreamBooth*). 93
62 SD v2.1 *Supplementary+AFFiNe* synthetic images (*LoRA*). 94
63 SD v2.1 *Supplementary+AFFiNe* synthetic images (*DreamBooth*). 95

List of Tables

1 Image augmentation choice per classifier architecture [6]. 12

2 Grid search for the best RandomAugment hyperparameters. 46
3 Grid search for the best AugMix hyperparameters. 47
4 Caption prefixes for fine-tuning datasets. 50
5 DreamBooth training settings. 51
6 Synthetic data generation final candidates. 53

7 Baseline models metrics. 56
8 RandAugment augmentation metrics on *ResNet18* model. 57
9 RandAugment augmentation metrics on *ResNet50* model. 57
10 AugMix augmentation metrics on *ResNet18* model. 58
11 AugMix augmentation metrics on *ResNet50* model. 59
12 Faster AA augmentation metrics on *ResNet18* model. 60
13 Faster AA augmentation metrics on *ResNet50* model. 60
14 Training with synthetic data on *ResNet18* model. 61
15 Training with synthetic data on *ResNet50* model. 61
16 Total metrics comparison on *ResNet18* model. 62
17 Total metrics comparison on *ResNet50* model. 62
18 Total time spent on each technique (*hours:minutes*). 63
19 Baseline models metrics. 64
20 Best achieved models metrics. 64

1. Introduction

The rapid expansion of human activities in underwater environments made it necessary to perform efficient and reliable monitoring of underwater ecosystems, with the accurate classification of underwater objects and, in particular, fish species playing a crucial role [1]. Supporting healthy and diverse fish population maintain ecological balance, food security, and support economic activities related to fishing and aquaculture. Monitoring and understanding fish behavior is vital for determining the impact of human-made structures, such as hydroelectric power plants, and implementing measures to mitigate potential adverse effects on aquatic life. Accurate classification of fish species enables researchers to collect reliable data and make informed decisions, benefiting numerous applications and stakeholders.

Current fish species classification methods often rely on a combination of machine learning algorithms and image recognition techniques. Among the most popular approaches for image classification are deep learning-based approaches, such as Convolutional Neural Networks (CNNs) [2, 3]. These models in their training process automatically infer various features and complex patterns from large datasets of labeled images, leading to accurate classification results. However, the performance of these classifiers depends heavily on the quality and quantity of the available training data. Underwater images are often subject to degraded quality and visibility due to factors such as light attenuation, scattering, color distortion, and noise [3, 4]. Moreover, collecting and annotating large-scale datasets of fish species is time-consuming, labor-hungry, expensive, and in some cases, physically challenging due to limited access to certain habitats or species.

There are various large good-quality datasets available, such as *ImageNet*[1], *Pascal*[2], *COCO*[3] and others; that can be used as a base in a transfer learning process. However, the statistical properties of such datasets might be very different from the domain of application. Taking into account the above, these limitations pose a significant challenge to the development of robust and accurate classifiers.

Image augmentation has been widely used, besides increasing models' robustness, to address data scarcity issues, by creating new synthetic training samples from existing data through various transformations, such as rotation, flipping, scaling, and various visual

[1] https://www.image-net.org/
[2] http://host.robots.ox.ac.uk/pascal/VOC/
[3] https://cocodataset.org/

adjustments. By creating a more diverse and robust dataset, image augmentation techniques have the potential to significantly improve the generalization capabilities of classifiers, leading to better performance on unseen data and reducing the risk of overfitting [5, 6, 7].

There are numerous image augmentation techniques available, ranging from relatively simple geometric and color-based transformations [5, 8] to more sophisticated approaches based on deep learning architectures like Generative Adversarial Networks (GANs) [9], Variational Autoencoders (VAEs) [10], and latent diffusion models [11], such as Stable Diffusion [12]. Each of these methods has its unique advantages and drawbacks, and their effectiveness in enhancing the performance of image classifiers may vary depending both on the training dataset and the domain of the task [6, 13]; and on the specific architecture of CNN (and not only) models (see Table 1).

Table 1. Image augmentation choice per classifier architecture [6].

Architecture	Proposed augmentation algorithms
AlexNet	Translate, Flip, Intensity Changing
ResNet	Crop, Flip
DenseNet	Flip, Crop, Translate
MobileNet	Crop, Elastic distortion
NasNet	Cutout, Crop, Flip
ResNeSt	AutoAugment, Mixup, Crop
DeiT	AutoAugmentat, RandAugment, Random Erasing, Mixup, CutMix
Swin Transformer	RandAugment, Mixup, CutMix, Random Erasing

Although various image augmentation techniques have been used for improving the performance of image classifier models, the impact of the application of these approaches on a specific case is unknown and is a subject to address. Moreover, with the recent boom of new generative models, such as Stable Diffusion [14, 15, 12], the comparison of novel approaches with the classic ones is also subject to address.

1.1 Related literature

The importance of image augmentation techniques in enhancing the performance of image classification models has been widely recognized for a long time. Several recent articles [16, 6, 7] provide comprehensive surveys of the various techniques available, ranging from basic transformations to more complex methods involving GANs. One survey present [16] the taxonomy of some augmentation techniques (including generative models) as well as evaluating the results of applying each technique on well-known open

datasets like *CIFAR-10*[4] and *CIFAR-100*[4] on the multiple Computer Vision (CV) tasks, including image classification as well as presenting possible challenges in achieving the best results. However, it is unknown if applying described approach will lead to similar model improvements in different conditions and it was not fully clear which augmentation technique was applied to get the particular result.

Later surveys [6, 7] expanded the taxonomy presenting and explaining the basics of a wide range of augmentation techniques, and, while authors in [6] did not provide any evaluations, [7] compared each[5] approach of various datasets, such as *CIFAR-10*, *CIFAR-100*, *ImageNet*; with *AutoAugment* and *RandAugment* based approaches achieving the stable and robust results in every tested dataset. Also, the same approaches (*AutoAugment* and *RandAugment*) were found in state-of-the-art solutions of *Fine-Grained Image Classification* tasks (at the state of 06-04-2023)[6].

The problems with the presented results are that they were achieved on big clean popular datasets, fully or partially not related to the classification of fish species, and, moreover, not covering the new possible image augmentation candidates, such as latent diffusion models.

Moving to the fish species image classification, in one article [3] authors proposed a classification method on custom CNN with some image thresholding and achieved an increase in $4,29\%$ of a classifier performance compared to the previous papers. They did not use any augmentation and their dataset contained 27k unbalanced image-label pairs. The fact that they have used the *accuracy* metric might gave the overestimated expectations due to possible overfitting. Authors of another article [17] compared the combination of various CNN models (*AlexNet, GoogLeNet, Caffenet and VGGNet*) and some basic augmentation techniques (Rotating, Mask) and achieved an 3% improvement of a combination of augmentation compared to the baseline models (with no augmentation). They have also used *accuracy* as a metric, but the application of augmentation might have saved the model from overfitting and they have also used *loss* value of a test dataset as a metric which also makes the result look more confident.

In one recent article [18] authors presented the approach of conditional augmentation based on the class loss gradient, applying augmentation on images of a class that has convergence problems. With this approach they have achieved up to 0.65% increase of *accuracy*, and up to 3.19% increase of *precision*. It should be noted that such small

[4] `https://www.cs.toronto.edu/~kriz/cifar.html`
[5] despite that GAN was presented, the evaluation of it was not demonstrated
[6] `https://paperswithcode.com/task/fine-grained-image-classification`

changes are related to the quality of baseline models. However, it was not clear, which particular augmentation techniques were applied conditionally, and it is also unknown how conditional augmentation might affect overfitting.

Fish species image classification remains a challenging topic due to factors such as underwater image quality and data scarcity. While several studies have explored this area, they tend to be limited to basic augmentation techniques. The performance improvements achieved through these techniques are often modest compared to the original model performance, making it difficult to distinguish between different augmentation techniques and identify the best one. Also, the choice of evaluation metrics is debatable, not accounting for class imbalance and possible overfitting. Moreover, the used datasets are quite large, in many cases, and the application of image classification and augmentation in limited-data scenarios needs to be investigated.

In one similar study [19], the author explored the use of GAN-based image augmentation to improve the classification performance of a CNN model for Parkinson's disease using a dataset of only 930 images on two classes. The approach resulted in a 2.3% increase in validation performance, while basic augmentation techniques led to a decrease in performance. The author has also provided deep analysis on different GAN-based approaches and the challenges that might occur in training and evaluation steps.

The recent boom of latent diffusion models has generated considerable interest in their potential use for image augmentation [14, 15]. These models are capable of producing photo-realistic images that, in some cases, can surpass the results achieved with GANs (see Figure 1). The application of these advanced models to fish species image classification could potentially yield significant improvements in model performance. However, training such models from scratch requires a dataset containing millions of images. To address that, the various fine-tuning techniques have been presented, such as *DreamBooth* [20] and adapted for image generation *LoRA* [21], allowing to transfer a new concept having only a few (3-5) images of a subject, with saved fidelity and ability to apply learned subject to a different context (i.e. changing the background). Moreover, some articles [22] already have suggested improvements to better control of a generative process, leading to better preservation of a target dataset domain, however, that might give the synthetic data a too high bias towards the original one.

Figure 1. Comparison of GAN-based augmentation to latent diffusion [23].

1.2 Problem statement

CNNs have been used for the classification of various objects, in particular fish species, their performance heavily relies on the availability of large, high-quality datasets. This is particularly challenging in the context of fish species classification, as underwater images often suffer from degraded quality and visibility. Although large open datasets could potentially be used as initial weights or on their own, they are too different in the domain of images. To address these problems, image augmentation has been used.

However, while much research was done on good-quality large datasets, it was not clear, how this would work in low-quality limited data scenarios, where alternative ways of collecting more real data are not financially profitable or too time-consuming, in particular, in terms of image classification model improvements. Also, new augmentation techniques, such as latent diffusion models, have not been extensively tested in the context of fish species classification, where basic augmentation techniques are still predominantly used. In addition, current research studies have often reported results in terms of accuracy metric, which are prone to data imbalances and may lead to overestimated results due to overfitting

To fill the gap, this study has investigated the impact of various image augmentation techniques on fish species classification in the described scenario, with a focus on evaluating and comparing the performance of traditional techniques and novel ones, such as Stable

Diffusion, in terms of both effects on a final classification model and costs of applying each one. Additionally, the study has explored the feasibility of applying these techniques to small initial datasets and will assess their effectiveness in mitigating issues related to dataset imbalances and overfitting.

The main goal of this work was, starting with a predefined small fish species dataset and a specific choice of CNN classification model architectures, to try to improve the performance of the baseline model by applying various augmentation techniques and reporting the results of their application. The compulsory goal was to answer the following **research questions**:

1. What are the specific combinations of augmentation techniques that yield the optimal performance improvements in a specific model?
2. Would it be possible to evaluate the effect of augmentation techniques application on a dataset in the context of a low-quality limited-data scenario?
3. Is novel argumentation techniques, such as diffusion models, can create synthetic data that lead to models outperforming ones trained with traditional augmentation methods?
4. In overall, can advanced image augmentation techniques be effectively applied in limited data scenarios?

1.3 Author contribution and scope

Given the vast range of image augmentation techniques and CNN models, it was not possible to cover all possible combinations in this study. Therefore, the scope of this work was limited to specific aspects of the problem:

- The author has focused primarily on well-known popular augmentation techniques, such as *AutoAugment*, *RandAugment*, and others; along with novel advanced techniques, such as Stable Diffusion. This study has helped to assess the effectiveness of state-of-the-art methods in the context of fish species multi-class classification in limited data scenarios, while also providing a comparison with traditional techniques.
- The author have considered a pair of popular CNN models — *ResNet18* and *ResNet50* with no pre-trained weights. This has limited the time and resources spent on model training and evaluation while providing sufficient information.

1.4 Thesis structure

The rest of this work is organized as follows:

Chapter 2 describes the theory and technical background of state-of-the-art augmentation techniques and other necessary information as well as provides justification for selected ones for the experiments.

Chapter 3 describes the data used in this study as well as the challenges and initial pre-processing necessary for the experiments.

Chapter 4 describes the experimental workflow and intermediate artifacts.

Chapter 5 presents the validation of achieved results.

Chapter 6 discusses the application of achieved results and contribution to further research as well as answers to the research questions.

2. Preliminaries

The process of CNN model training can be segmented into various parts:

1. Data acquisition
2. Data prepossessing
3. Model architecture selection
4. Data augmentation
5. Model training
6. Model evaluation

This chapter aims to provide necessary technical background and description necessary to understand the following chapters. *Data acquisition* and *initial prepossessing* parts are described in a separate chapter (see Chapter 3).

2.1 CNN model selection

CNN are a class of deep learning models specifically designed for processing large grid-like data, such as images. Unlike traditional Artificial Neural Networks (ANNs), which have full connections between each neuron (see figure 2), CNN extends that by convolutions (as well as other techniques). Within that, they are widely used in various CV tasks like image classification, object detection, and semantic segmentation [24].

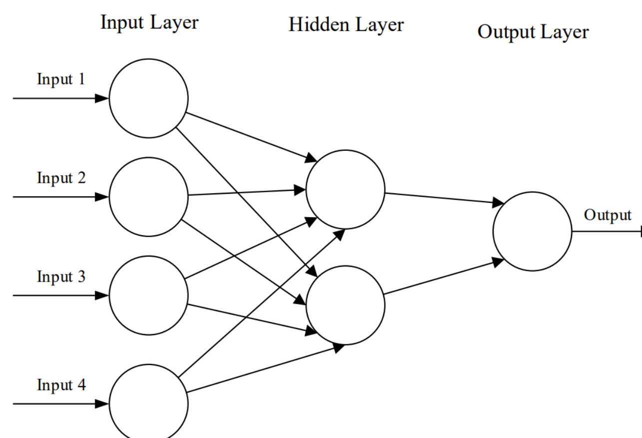


Figure 2. ANN structure [24].

As the name suggests the main part of a CNN is convolutional layers. These layers are the core building blocks of CNNs. They perform convolution operations on the input data

using a set of learnable filters (or kernels). Each filter is applied to a local region (window) of the input and produces a feature map (see Figure 3). By learning different filters, the convolutional layer can capture various spatial features like edges, textures, and shapes.

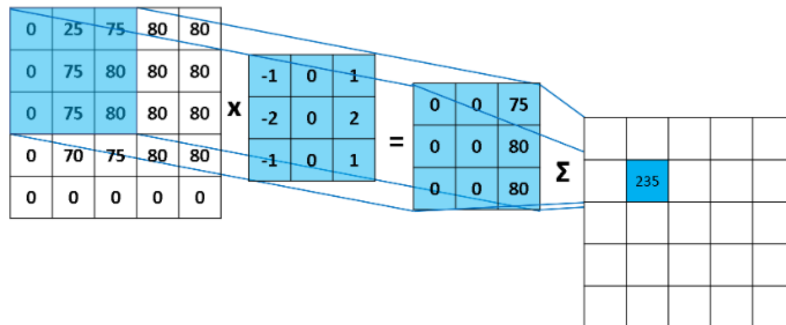


Figure 3. Principe of convolutional layer[1].

A typical CNN is composed of multiple layers, including convolutional layers, pooling layers (that reduce the size of the matrix by taking either average (avg-pooling) or max (max-pooling) value of a window), and fully connected layers. Each layer is responsible for learning and extracting different levels of features from the input data. The general structure of a CNN can be summarized as follows (see Figure 4).

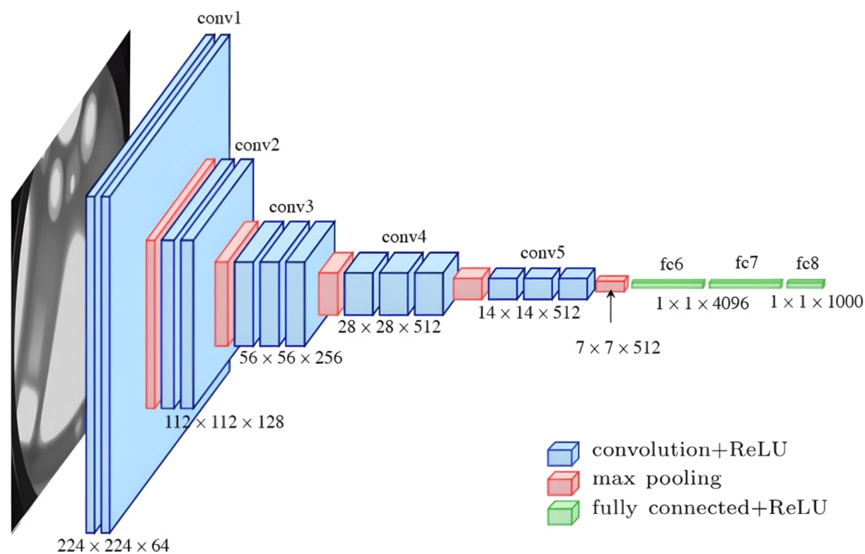


Figure 4. Typical CNN structure[2].

Typically at the end of CNN there is a fully connected layer and in the case of image classification, it has as many output neurons as there are classes.

[1]https://mlnotebook.github.io/post/CNN1/
[2]https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

2.1.1 ResNet

One of the main problems of the standard CNNs is that with the increase of the number of the layers aiming to make a more robust model, at some point a model becomes such deep, that weights updates (gradients) disappear in a propagation through the net in a process called *back-propagation*. The problem of vanishing gradients not only doesn't help the model to generalize input data but also leads to degradation in accuracy as the network grows deeper. To address that a concept of residual connections (see Figure 5) was introduced which enables more efficient training of deeper networks [25].

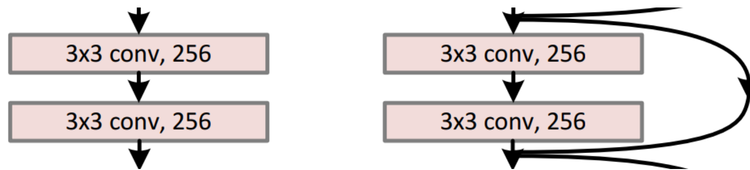


Figure 5. Comparison of plain CNN and ResNet layers [25].

A block with residual connection acts very similar to the plain CNN one, the only difference of combining the output of the current convolutional layer block $F(x)$ with the output of the previous block x producing the final output $H(x)$ (see Figure 6).

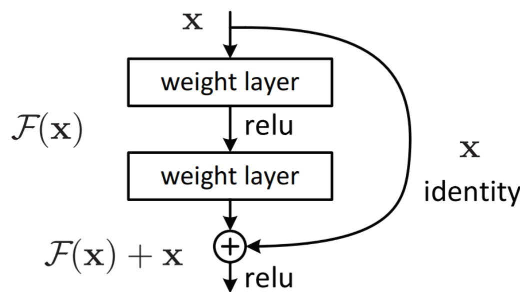


Figure 6. A block with residual connection [25].

ResNet was presented in various depths, such as *ResNet-18*, *ResNet-34*, *ResNet-50*, *ResNet-101*, and *ResNet-152*, where the numbers indicate the total number of layers in the network. While the **ResNet-18** has similar performance compared to the plain CNN with the same amount of layers, **ResNet-50** shows the increase in performance due to the residual connection. With a layer size of 152, ResNet managed to achieve 3.57% top-5 error on the ImageNet, having better performance and a much less complex structure than for example VGG model.

In this study results of the augmentation application were done on **ResNet-18** and **ResNet-50**. While ResNet-30 performs better than *ResNet-18*, it has more layers, so *ResNet-18* was selected as the simplest one, and *ResNet-50* as more complex and hence robust, while being relatively fast to train.

2.2 Data augmentation pipeline

By default CNN are bad at learning the invariance of unseen data, which is especially a problem with small datasets [26]. This can be fixed by either providing more training examples (i.e. collecting more data, which is not the case for this study) or making some changes to the training data to help a model generalize it. This is what data augmentation is used for.

There are differing opinions on whether to apply augmentation to only the training subset or the entire dataset. The most common approach is to apply it to the training subset (see Figures 7–8), allowing the model to learn from diverse examples while testing on original data. The alternative approach makes the test data look similar to the training data, which can help the model when augmentations significantly alter the data.

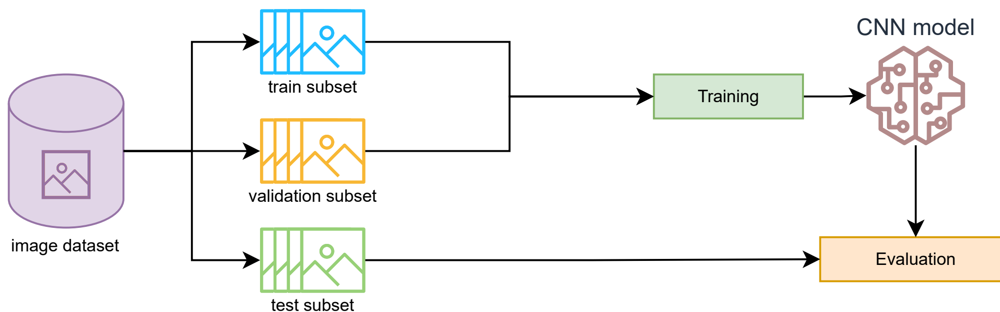


Figure 7. CNN model training pipeline without augmentation.

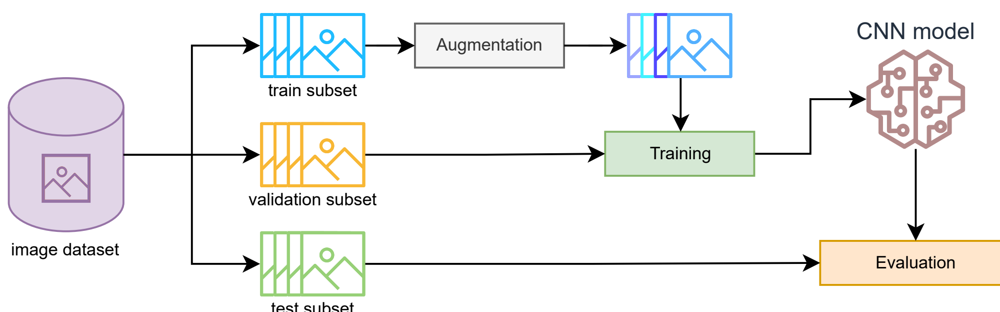


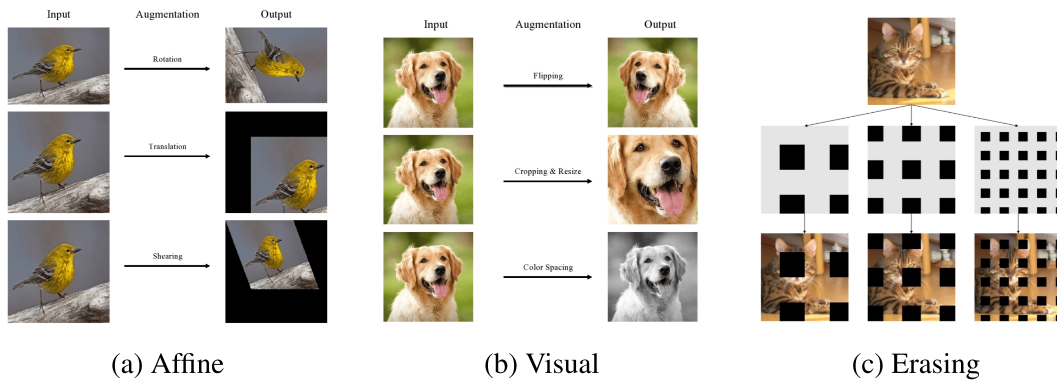
Figure 8. CNN model training pipeline with augmentation.

In this study, the augmentations were applied only to the training dataset, to see how the model will perform in general cases (unmodified evaluation images). Also in terms of the generative augmentation techniques, synthetic images were used only as additional training samples.

2.3 Basic augmentation

Basic data augmentation techniques are (typically) simple functions that are applied to the input images. They can be segmented into the following categories (see also Figure 9):

- Geometric augmentations (a.k.a. affine transformations)
- Non-geometric augmentations (visual transformations)
- Erasing augmentations



(a) Affine (b) Visual (c) Erasing

Figure 9. Examples of basic augmentation techniques [7].

Geometric augmentations are used to teach a model positional invariance and among the most common transformations are rotation, translation, and shearing. *Visual augmentations* gives makes a model robust to various positional visual invariances, among the most common are: flipping, color-spacing (grayscale) cropping, and resizing. *Erasing augmentation* share the same idea as *dropout* [27], removing some parts of the image to prevent individual parts of a model to rely only on those parts [7].

These basic augmentation techniques can be combined in various ways to create a more diverse and robust training dataset. However not all augmentations are suitable for particular datasets, for example, flipping and rotation are not suitable for the famous digits dataset *MNIST*, due to indistinguishability between rotated digits 6 and 9 [5].

2.4 Advanced augmentation

One of the main problems of basic augmentation techniques is to find out which transformations to use, with which probability, order, and other hyperparameters. Advanced augmentation techniques address that problem by introducing an optimization process of finding the most useful basic augmentations and their combinations.

2.4.1 AutoAugment

AutoAugment (AA) is an advanced data augmentation technique proposed by researchers at Google Brain in 2019 [28]. It is designed to automatically discover and apply the best augmentation policies to improve the performance of deep learning models for image classification tasks. The key idea behind AA is to use a search algorithm, that utilizes Reinforcement Learning (RL) to explore the space of possible augmentation policies and find the most effective ones for a given dataset and model.

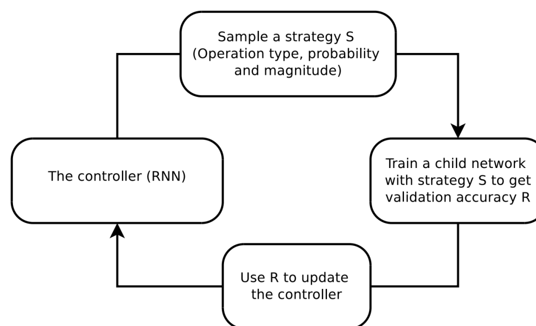


Figure 10. AutoAugment best policy search optimization cycle [28].

The *search space* in AA is defined as the set of possible basic augmentation *operations* (rotation, flipping, translation, shearing, etc.) and their corresponding ranges for parameters (angle of rotation, translation factor, etc.). A *policy* is a sequence of augmentation operations with their associated probabilities and magnitudes. In AA, a policy consists of 5 sub-policies and each sub-policy includes two operations. The algorithm learns the optimal policy by searching through the space of possible policies.

AA automatically discovers the best augmentation policy for a specific dataset, eliminating manual tuning of parameters and operations. It has been shown to improve the performance of deep learning models on various image classification datasets such as *CIFAR-10*, *CIFAR-100*, *SVHN*[3], and *ImageNet* [29]. However, AA requires additional controller training in a very big search space (AA with 5 sub-policies has $(16 \cdot 10 \cdot 11)^{10} \approx 2.9 \cdot 10^{32}$ possibilities), which increases the time and computational resources and just not worth training for testing purposes. This problem can be solved by using policies found by AA for other datasets (AutoAugment-transfer) [28], but there is no guarantee that a specific policy will work well for all datasets.

AA implementation is built into popular machine learning frameworks like Pytorch, with predefined policies for *ImageNet*, *CIFAR-10*, and *SVHN* datasets[4].

[3] `https://paperswithcode.com/dataset/svhn`
[4] `https://pytorch.org/vision/main/generated/torchvision.transforms.AutoAugment.html#torchvision.transforms.AutoAugment`

2.4.2 Fast and Faster AutoAugment

Proposed at NAVER AI Lab in 2019 [30] **Fast AA** aims to reduce the high computational cost and complexity of AA while maintaining compatible effective augmentation policies. Fast AA utilizes a more efficient search strategy by replacing the RL-based controller in AA with a more efficient Bayesian optimization approach and distributed training.

In Fast AA, the dataset is divided into K equal-sized subsets (folds), each containing a pair of D_M and D_A subsets. Then a classification model parameter θ is trained on a D_M. After that, a bundle of suggested policies is obtained by the Bayesian optimization algorithm and evaluated on a D_A, with top N policies beige added to a global set of found policies (see Figure 11).

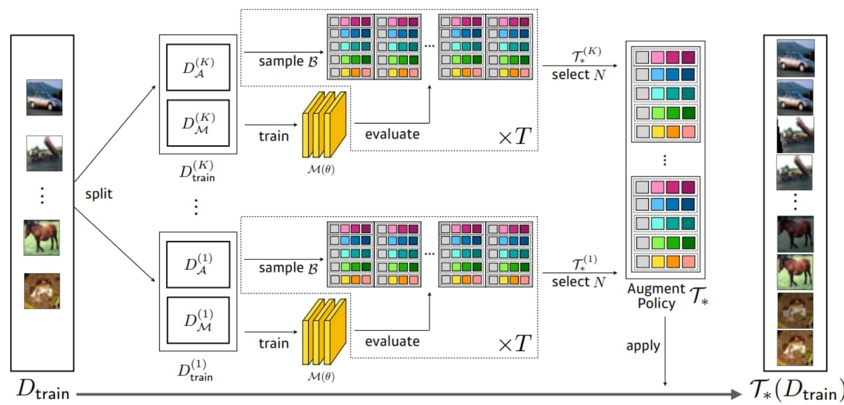


Figure 11. An procedure of best policies search in Fast AA [30].

However in their paper authors proposed a Fast AA version that utilizes distributed learning, which makes it hard to run for a particular dataset on a home machine.

Proposed by RIKEN in 2019 [31], **Faster AA** achieves even faster policy searching for data augmentation than AA and Fast AA without a significant performance drop. The two core concepts of Faster AA are fully differentiable basic data augmentations, that allow one to train a model to predict best augmentation policies via back-propagation; and a modified GAN where instead of a traditional generator G and discriminator D, G applies a specific augmentation and D tries to guess if this was a real image or an augmented one (see Figure 12).

Unlike Fast AA Faster AA does not require distributed training and can be easily trained on a home machine. In particular a ready-to-use solution was implemented for a popular python augmentation library *Albumentations* [5] — *AutoAlbument*[5].

[5]https://albumentations.ai/docs/autoalbument/

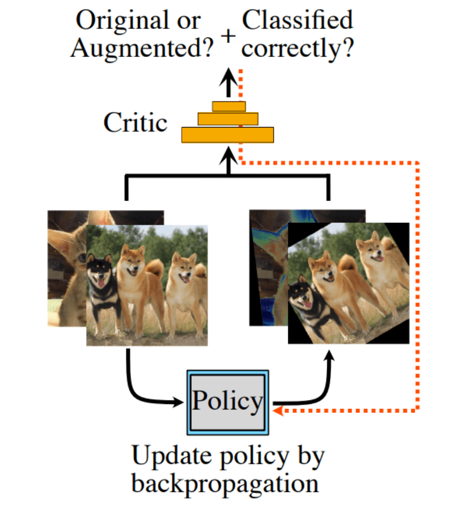


Figure 12. Differentiable data augmentation training in Faster AA [31].

2.4.3 RandAugment

RandAugment (RA) is a data augmentation technique introduced by researchers at Google Brain in 2019 [29] as a more efficient and simpler alternative to AA and Fast AA. RA is a ready-to-use advanced augmentation technique that, unlike AA, Fast AA, and Faster AA does not require any complex search algorithms or additional training phases.

RA has two main hyperparameters: N, representing the number of augmentation operations per image, and M, controlling the global magnitude of the applied operations. With only 2 hyperparameters the search space is dramatically reduced, which allows one to find optimal hyperparameters using brute force (grid search). Given all possible basic augmentations, a tuple of N is selected with a uniform distribution and then applied to the input image with M magnitude (see figure 13).

The drawback of this approach is a stochastic behavior due to a random aspect of basic augmentations selection, which might require several training cycles of a target model to get an objective evaluation of selected N and M hyperparameters.

Implementation of RA has been build-in into Pytorch framework[6].

[6] `https://pytorch.org/vision/main/generated/torchvision.transforms.RandAugment.html#torchvision.transforms.RandAugment`

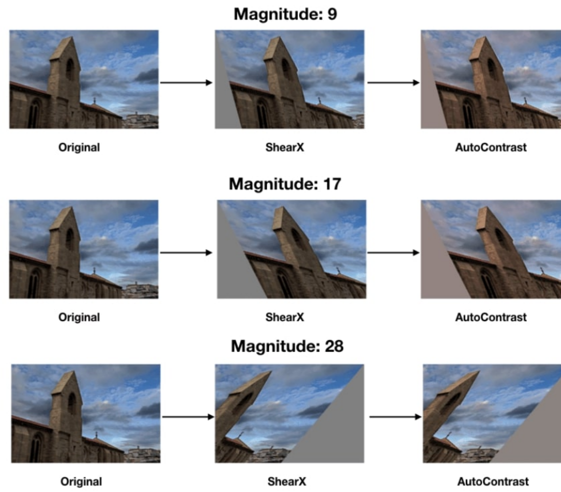


Figure 13. Example images augmented by RandAugment [29].

2.4.4 AugMix

AugMix is a data augmentation technique proposed by researchers at Google Research in 2020 [32] to improve the robustness and generalization of deep learning models. Like RA AugMix utilizes a stochastic approach of selecting a group of basic augmentation operations (chains). Then a mixing weight is assigned to each chain forming a complex augmentation pipeline. After that each image is merged with the same image but after passing this pipeline resulting in a slightly different image, which is enough for creating a diverse and robust to image distortions augmented dataset (see figure 14).

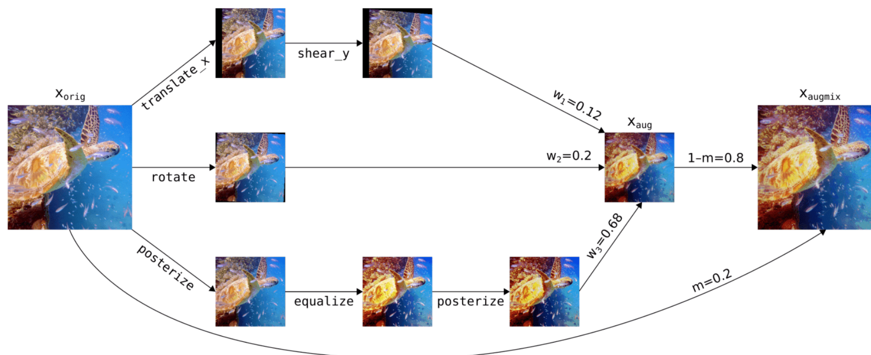


Figure 14. Example of AugMix application on an image [32].

AugMix introduces additional hyperparameters, such as the number of augmented versions to blend and the strength of consistency regularization. Tuning these hyperparameters may require additional computational resources and experimentation.

Implementation of AugMix has been build-in into Pytorch framework[7].

[7] https://pytorch.org/vision/main/generated/torchvision.transforms.

2.5 Stable Diffusion

There is some limitation of using basic transformations (or their combination in terms of advanced augmentation techniques): the augmented data is still very similar to the original images, while such transformations are likely to affect the image itself rather than the object on that image.

Stable Diffusion (SD) presented by Stability Ai is a novel text-to-image generative model guided by the user text descriptions (**prompts**). It is a complex model consisting of the following main components [12]:

- Text encoder
- Diffusion model (UNet + Sampler)
- Image decoder

Text encoder is part of a large language model from the Natural Language Supervision domain [33]. The general idea is by utilizing the text encoder and image encoder (both convert initial data into a relatively small vector space) to find a match between textual description and input image (see Figure 15). Stable diffusion uses that text encoder to transform user prompts into vectors that can be used to guide the image generator model.

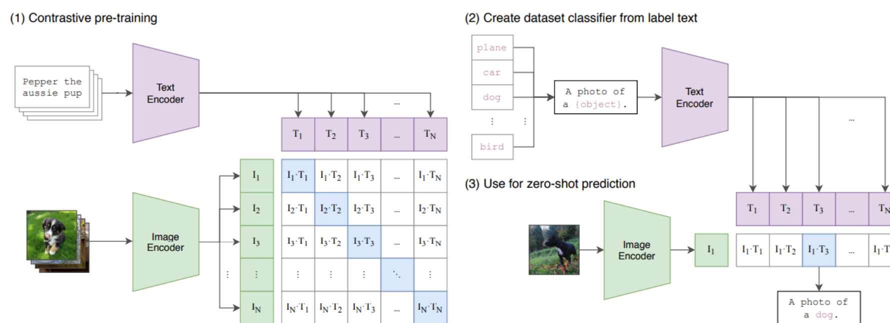


Figure 15. Example of AugMix application on an image [33].

Diffusion models (or latent diffusion models) are a class of generative models that have recently gained attention for their ability to generate high-quality images. While they have primarily been used for unconditional image synthesis, with guidance from a prompt, they can be used to generate specific images with any context. Under the hood, they are Markov chain models that are learned to recreate images from noise by step-by-step noise removal (the reverse diffusion process) [11] (see Figure 16).

Diffusion models by themselves consist of two major components: a U-Net backbone

AugMix.html#torchvision.transforms.AugMix

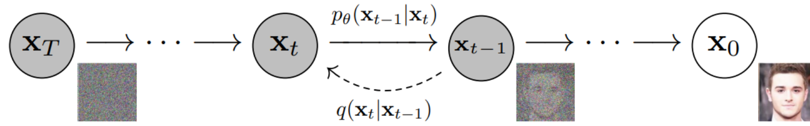


Figure 16. Example of AugMix application on an image [11].

and a sampler. While samplers are responsible for defining the amount of noise removal schedule, the U-Net backbone is responsible for the reverse diffusion process. In Stable diffusion, U-Net was updated to add attention layers that guide the denoising process by text description vectors (see Figure 17).

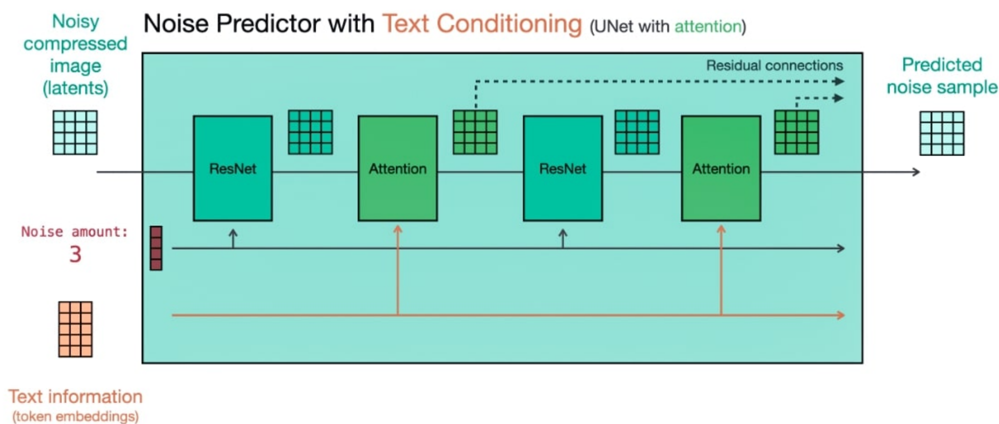


Figure 17. U-Net noise predictor with text attention layers[8].

The last part of Stable diffusion — *Image decoder* is responsible for converting an image in latent space to a real image in pixel space. Initially, diffusion models were used on bare images, which make both the training and evaluating processes extremely slow [12].

2.5.1 Stable Diffusion v1.5 and v2.1

With the release of their paper, Stability AI also releases first v1.x (*v1.5*) model[9] and then v2.x model[10] both under **CreativeML Open RAIL-M** licence[11] that allows for both commercial and non-commercial usage.

[8] https://jalammar.github.io/illustrated-stable-diffusion/
[9] https://stability.ai/blog/stable-diffusion-public-release
[10] https://stability.ai/blog/stablediffusion2-1-release7-dec-2022
[11] https://huggingface.co/spaces/CompVis/stable-diffusion-license

2.5.2 Fine-tuning: DreamBooth

While Stable Diffusion gives large possibilities of what it can produce just by using some text description, the domain of produced images though depends on training data that was given to it. Trained on *LAION* dataset[12] plain Stable Diffusion cannot produce images of a correct fish specie suitable for this work (see Figure 18).

(a) Generated (b) Real [34]

Figure 18. Comparison of "*photo if an eel*" generated by SD and a real photo.

In their paper [20] authors present a novel approach to fine-tuning the Stable Diffusion model with a few (typically 3–5) images of a subject. They did this by freezing the initial model and training its copy. Using a unique identifier (in the paper authors suggest using rare word up to 3 symbols) and *class preservation loss* they fine-tuned the model in the way that the image produced by the trained model of the original class stays close to that generated by the frozen model, while image generated with identifier stays closer to the train images (see Figure 19). This makes a fine-tuned model more robust to overfitting (which is important as the train set is very-very small).

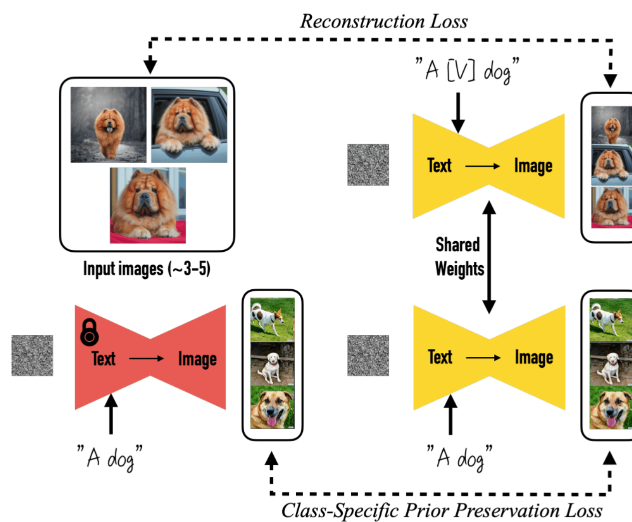


Figure 19. Fine-tuning Stable Diffusion using DreamBooth.

[12] https://laion.ai/

There are plenty of training scripts, utilizing *DreamBooth*, in particular for this study *fast-Dreambooth* colab notebook was used[13].

2.5.3 Fine-tuning: LoRA

There are a few problems with *DreamBooth*: it is still time-consuming to train, especially in cases where the fine-tuning dataset is more than a few images; and the resulting model is as big as the initial model which is problematic for experimentation purposes.

Initially released for Large Language Model finetuning, *LoRA* utilizes the approach of low-rank update matrices. In simple words, instead of fine-tuning the entire model authors of [21] proposed a way of training small additional weights ΔW decomposed by Singular Value Decomposition (SVD) into a low-rank format.

This idea was further adapted by [35] for Stable Diffusion fine-tuning. This gives an end user the ability to train a subject onto a pre-trained Stable Diffusion model even faster than *DreamBooth* (because only a low rand additional model in being trained), and resulting models are consuming very little space to store. Additionally, these models can be further controlled by modifying the merge constant α, which can reduce the effect of *LoRA* in case of overfitting and also allows to combine different *LoRA* models together to create an infinite amount of generative models (see equation 2.1).

$$W' = W + \alpha \Delta W \tag{2.1}$$

For this study the script by Kohya's was used[14]

2.6 Model training

CNN model training (just like general model training) does not always succeed, in particular, a model suffers from two most common problems: **Overfitting** and **Underfitting**. While underfitting is just a general artifact of undertrained models when either model structure, training data, or the whole process together does not lead to a good performance of an end model. Unlike underfitting, overfitting creates more confusion for model evaluation because, from the perspective of training data, the model performs well, but is failing on previously unseen data because of the memorization of training data instead of

[13] `https://github.com/TheLastBen/fast-stable-diffusion`
[14] `https://github.com/bmaltais/kohya_ss`

generalization (see Figure 20) [36]. Data augmentation might help solve both of these problems, but might also make training slower, which under the same number of steps might lead to underfitting compared to other models.

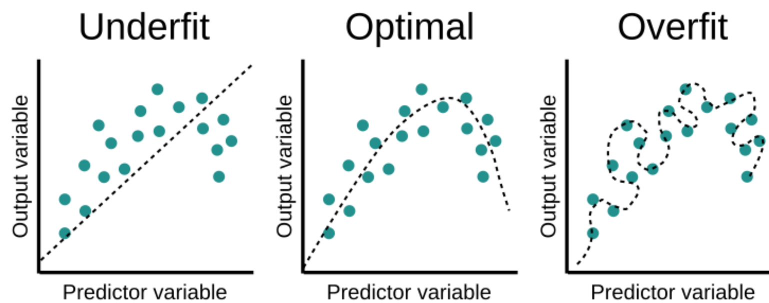


Figure 20. Example of Underfitting and Overfitting[15].

Another problem is **Data Leakage** which can lead to a wrong performance evaluation because some of *previously unseen* data was not fully unseen but actually mixed with training samples. To solve that, with regards to augmentation, test data should be separated from train data before the augmentation step [37].

2.7 Validation

To evaluate trained models and, in particular, evaluate the effect of specific augmentation techniques, a model must be tested under different evaluation metrics. This section presents validation techniques used in this work to evaluate and analyze the results.

Cross-validation. Not a metric but rather an approach to getting more statistically confident results by training a model several times on different subsets of data (folds) [38].

Confusion matrix. A useful way of representing predicted values of true values (see Figure 21), especially in terms of multi-class classification which in this work is fish-species classification. Closely related definitions are **True Positive (TP)**, **True Negative (TN)**, **False Positive (FP)**, **False Negative (FN)**. TP and FP represent the number of correct and incorrect predictions of a specific class while TN and FN represent the number of correct and incorrect predictions of any other class [39].

Accuracy shows the general performance of models (total proportion of correctly predicted labels over all predictions). Ideally, this value should be close to 1 (or to 100%). The drawback of this metric is that it is not agnostic to class imbalance, say, having good predictions for a major class accuracy might show that model performs almost 100%, but

[15]https://www.fastaireference.com/overfitting

True class

	p	n
Y	True Positives	False Positives
N	False Negatives	True Negatives

Hypothesized class

Figure 21. Example of confusion matrix [39].

in reality, the model just picks that major class for all of its predictions. See the formula in equation 2.2.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

To give a more confident evaluation there are **Precision**, **Recall** (Sensitivity), and **Specificity** (true negative rate) (see equations 2.3-2.5).

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

$$TNR = \frac{TN}{TN + FP} \tag{2.5}$$

F1-score was designed to combine Precision and Recall metrics together into a single one. F1-score formula is defined as follows (see equation 2.6):

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.6}$$

3. Datasets

As was stated in Chapter 1, this study was investigating the application of various image augmentation techniques in low-quality limited data scenarios, where collecting more data of the target domain is not for some reasons suitable. In this chapter, two datasets are presented: *Supplementary* is a target dataset of the domain of interest, while *AFFiNe* is an additional dataset legally obtained from the internet to be used with generative models fine-tuning.

3.1 Supplementary

Supplementary [40] is a dataset used in this study for baseline model training and further augmentation techniques evaluation. This is a private dataset, and the right to use it was kindly given by The German Federal Institute of Hydrology (BfG)[1] as this study is part of a collaboration project between a research group at Taltech and BfG. This dataset contains **50 videos** of **10 different fish species**, that were automatically detected[2] at various fish counters at Koblenz, Germany. Each video has one specific label attached to it (i.e. each video presumable has only one kind of fish). Examples of video frames can be seen in Figure 22.

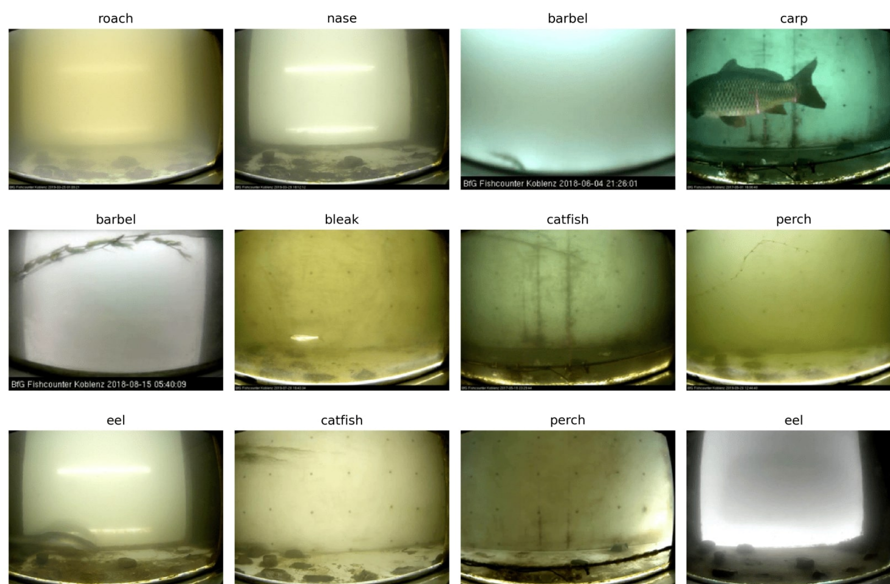


Figure 22. *Supplementary* dataset raw frames examples [40].

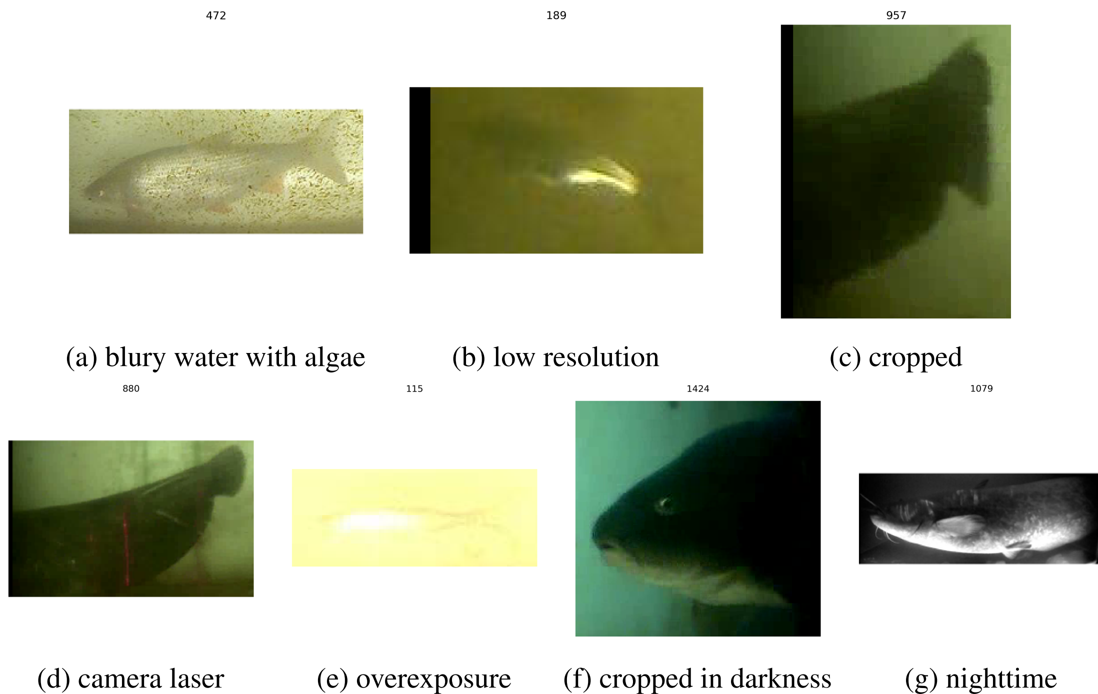
[1] https://www.bafg.de/EN/03_The_%20BfG/the_bfg_node.html

[2] Note: as this study was investigating the limited-data scenarios, the real possibility of obtaining more data was not considered

3.1.1 Challenges

Because this dataset represents underwater fish images captured through the glass of fish counters, it has various visual aspects, that might have been challenging for the model, in particular (see also the examples in Figure 23):

- Some videos have low resolution
- Some videos are taken in darkness
- Algae growing on a glass
- Some videos are mostly empty with just a few frames containing fish
- There is little chance of fish appearance of a different species, than those with which the video is labeled
- There is a certain imbalance of fish species frames (see Figure 26)
- Some video frames are overexposed
- Some video frames have blurry water
- Noise
- Fish are swimming, hence the full body of fish is not always captioned
- Some fish species are too big (catfish) to fully fit in the frame



(a) blury water with algae (b) low resolution (c) cropped

(d) camera laser (e) overexposure (f) cropped in darkness (g) nighttime

Figure 23. Examples of challenges with *Supplementary* dataset [40].

3.1.2 Preprocessing

To work with images, they have to be extracted from the videos first. To do so each video was processed using python library *OpenCV* and frame-by-frame saved in sub-directories according to the corresponding label of that video. Because frames are mostly similar, which brings no use neither for model training, frame labeling, and checking, each frame was compared automatically with the previous one, and if the difference was beyond the specific threshold, the frame was extracted, otherwise skipped. The distribution of labels and overall frames number can be seen in Figure 24.

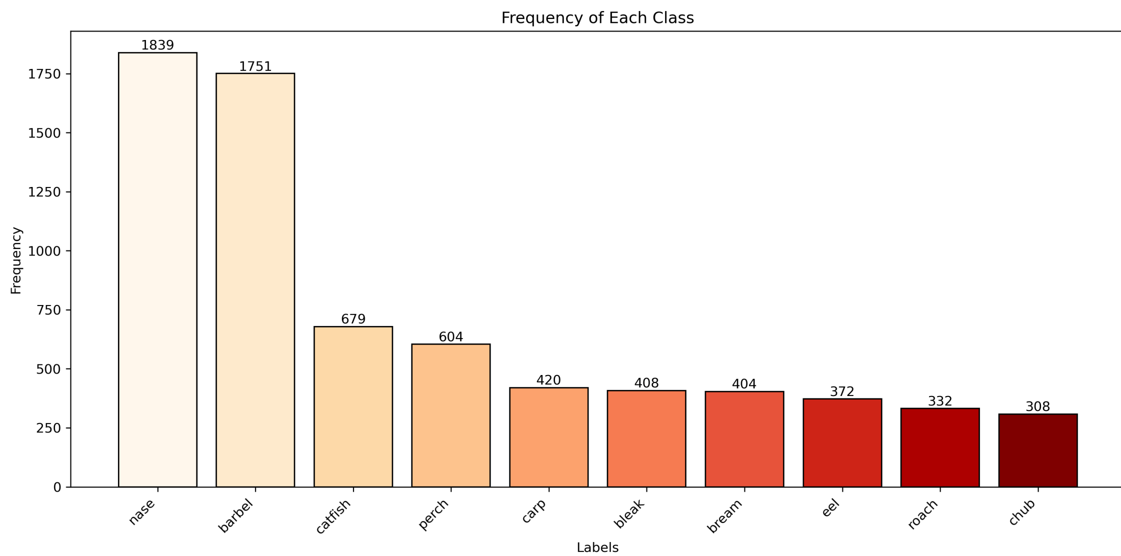


Figure 24. *Supplementary* dataset distribution after automatic extraction [40].

However automatic extraction was not enough to work with this dataset. After the extraction process, most frames were still empty and fish in most cases has covered only a small part of a frame. Next, each frame was manually checked and labeled with a corresponding bounding box around the fish. This was done using a custom Python Jupyter Notebook widget (see Appendix 2) with a help of a pre-trained fish detection model (You only look once (YOLO)).

After manually filtrating **7117** frames (removing frames with no fish, extracting bounding box around the fish, etc), only **2771** of them were left for further use. Filtered frames then were cropped with respect to bounding boxes and saved separately. Examples of filtered dataset frames and new label distribution can be seen in Figures 25–26.

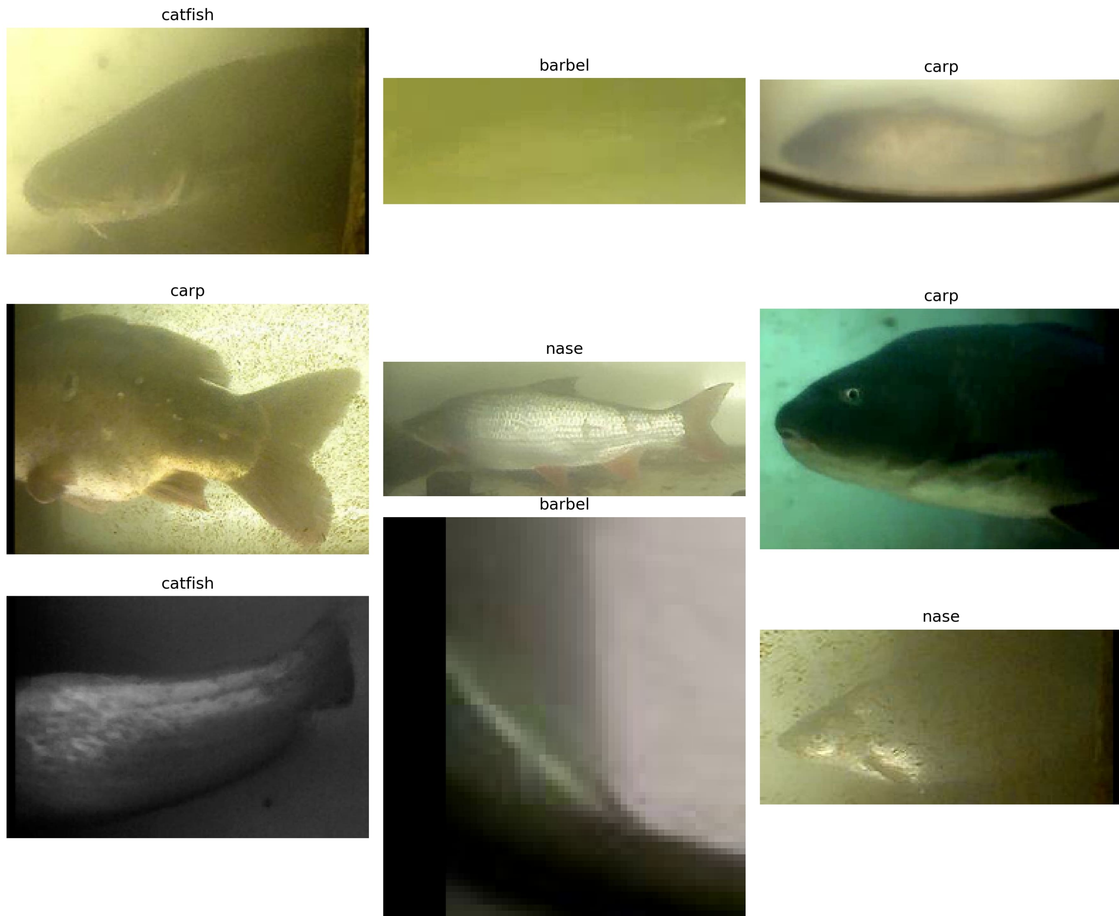


Figure 25. *Supplementary* dataset frames examples after prepossessing [40].

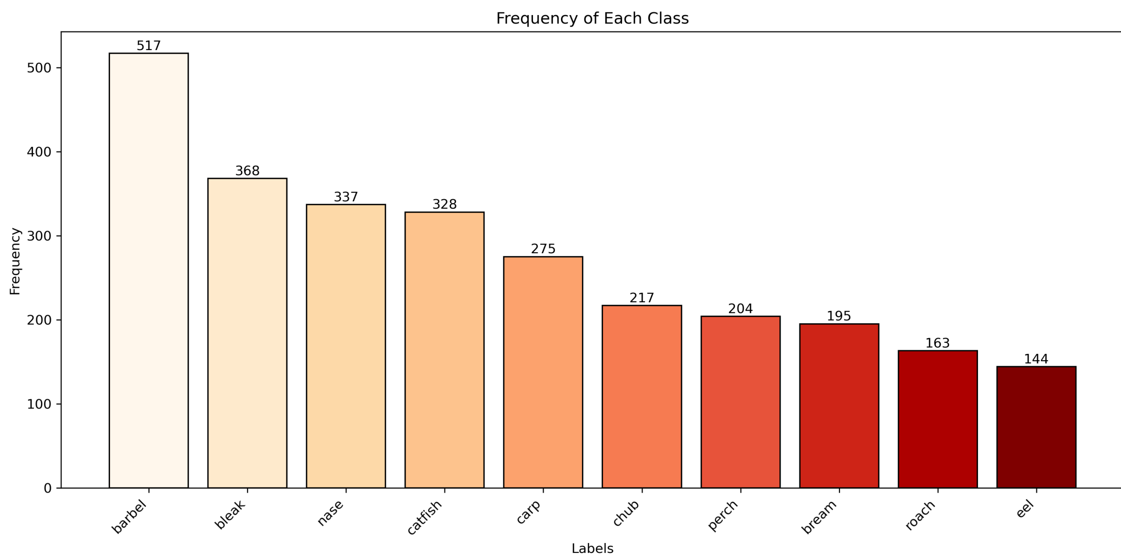


Figure 26. *Supplementary* dataset distribution after prepossessing [40].

3.2 AFFiNe

AFFiNe [34] is a publicly available[3] collection of over **7k** images of **30 freshwater fish species** found in the Netherlands. This dataset is designed to help in the development of machine-learning models for image classification and object detection in the context of fish species. The dataset has been published under a "**CC BY-NC-SA 4.0**" license[4], making it suitable for non-commercial applications such as this study. Examples of *AFFiNe* dataset frames and labels distribution can be seen in Figures 27–28

Figure 27. *AFFiNE* dataset raw frames examples [34].

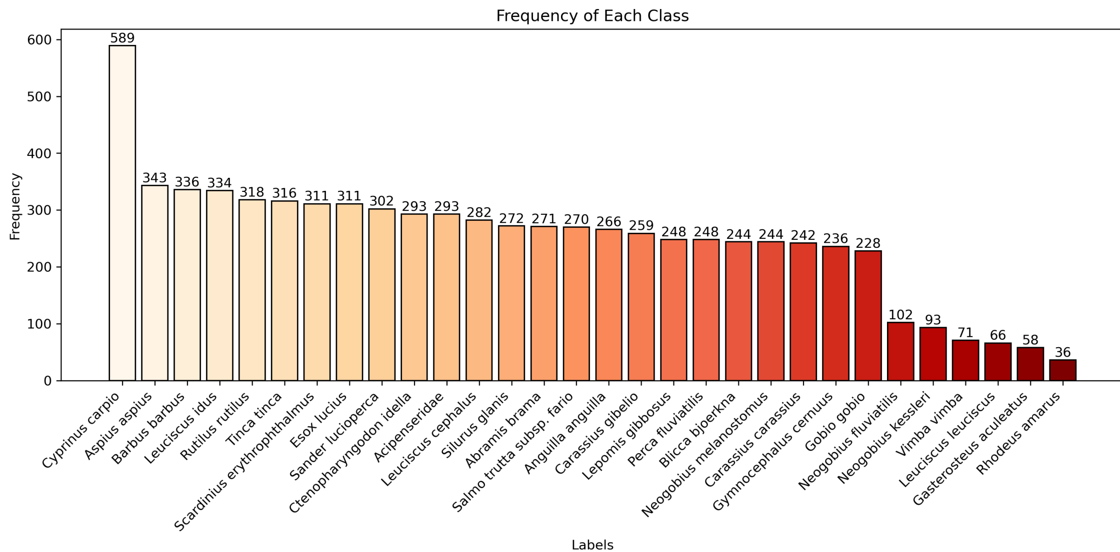


Figure 28. *AFFiNE* dataset distribution [34].

As the training process of generative augmentation techniques is by itself data consuming process, and the quality of the data matters a lot, the use of this dataset aims to improve the

[3] `https://www.kaggle.com/datasets/jorritvenema/affine`
[4] https://vaartsoftware.nl/beeldherkenning-nederlandse-vissoorten-openbare-dataset/

performance of these techniques trained on *Supplementary* dataset by introducing more data of the similar domain. However, since datasets are still fundamentally different it is necessary to compare the effect of generative augmentation techniques fine-tuned on **just *Supplementary*** with the effect of those that were fine-tuned on a **combination of *Supplementary* and *AFFiNe* datasets**.

As this dataset is already cropped to have fish in the center of the frame, no additional prepossessing was necessary at this step. Note: model training-specific particular transformations are not covered in this chapter and will be explained in the following chapter in detail.

Unlike *Supplementary*, *AFFiNe* has fish taken out of the water, moreover some images have fish being held by a human, which is another factor, which might make model training more challenging. On the other side, having fish in various conditions might help the model catch the necessary invariances to successfully classify fish species.

Also, both *AFFiNe* and *Supplementary* have an overlap of fish labels with *AFFiNe* having almost all species that are present in *Supplementary*, except for bleak (*Alburnus alburnus*) and nase (*Chondrostoma nasus*). From one perspective this might help the model trained on both datasets generate more *correct* species, but the lack of presence of two species might introduce another point of class imbalance.

4. Experiments

In this chapter, the experimentation workflow is described, including test condition setup, model-specific image transformation, and application of each augmentation technique discussed in this study.

4.1 Experimental setup

All experiments were done in Python using the popular deep learning library PyTorch[1].

As was previously described in Chapter 2 two models were used for the evaluation of augmentation techniques: **ResNet18** and **ResNet50**. Both models (as well as models with more layers) were obtained from *torch.hub* in a not pre-trained format (see Listing 4.1):

Listing 4.1. ResNet18 and ResNet50 loading code.

```
torch.hub.load('pytorch/vision:v0.10.0', resnet18, weights=None)
torch.hub.load('pytorch/vision:v0.10.0', resnet50, weights=None)
```

To make all experiments fair for every augmentation technique the following additional conditions were setup (see Listing 4.2):

Listing 4.2. Model training configuration.

```
BATCH_SIZE = 32
MAX_EPOCH = 100
NUM_WORKERS = 24
LR = 1e-1
loss_fn = torch.nn.CrossEntropyLoss
optimizer = torch.optim.Adam(...)
scheduler = torch.optim.lr\_scheduler.CosineAnnealingLR(...)
```

Almost every computation was done on a TalTech: Environmental Sensing and Intelligence group private development server using **NVIDIA GeForce RTX 3080 Ti** with 12 GB of VRAM.

[1] https://pytorch.org/

4.2 Data and model preparation

Before anything could be done it was necessary to both prepare data for training and the models.

4.2.1 Dataset

As was previously described in Chapter 3, *Supplementary* dataset consist of **10** unique fish species and **5** videos per each specie (50 in total). After extracting and filtrating images the dataset of over **2k images** was divided into two main subsets: the training subset and the test subset with the proportion of 4 train+validation videos per class and 1 for testing. Testing videos were manually selected with as best quality (resolution, lighting conditions, etc) as possible (see Listing 4.3 and Figure 29) and that was not uncompromising:

Listing 4.3. Test videos.

```
test_videos = [
    "original/eel/20190410_211027562.avi",
    "original/nase/20190329_181208678.avi",
    "original/perch/20171010_090409551.avi",
    "original/roach/20171008_194306707.avi",
    "original/barbel/20190930_155816446.avi",
    "original/bleak/20170814_060813737.avi",
    "original/bream/20190710_155802204.avi",
    "original/carp/20190613_104052513.avi",
    "original/catfish/20190627_015724073.avi",
    "original/chub/20171006_191548412.avi",
]
```

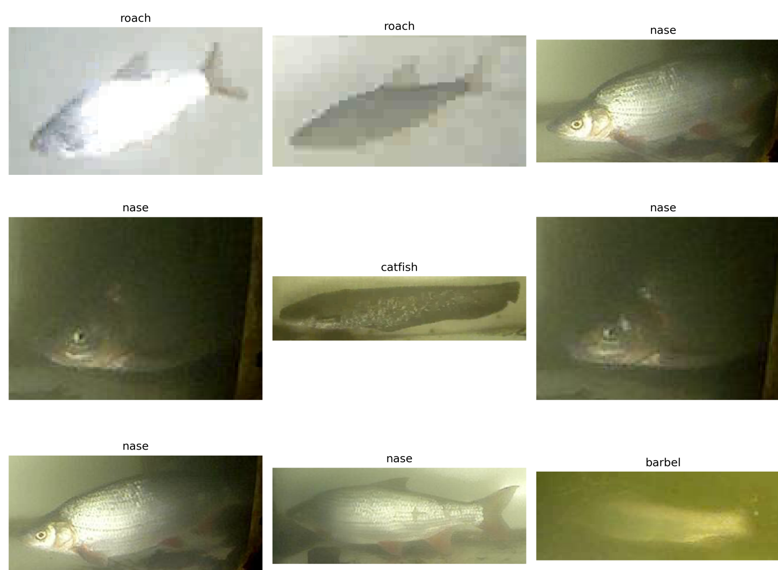


Figure 29. Example of test subset images [40].

This division is essential for evaluating the performance of the image classification model with data leakage as minimal as possible. Next the five pairs of *train+val* subsets were created:

- **4vs1** (rest of images split into 70% train and 30% validation data)
- **3+1vs1** (per each class 3 non-test random videos for train and 1 video for validation)
  - Subset A
  - Subset B
  - Subset C
  - Subset D

The purpose of *4vs1* is to demonstrate data leakage and its consequences on training and evaluation, and *3+1vs1* to train models in cross-validation style minimizing the data leakage. Unlike *4vs1*, videos of subsets *3+1vs1* do not overlap between train and validation data. These splits (into test and train+validation) were done once and each further model+augmentation technique was trained on the same data to make the comparison fair.

4.2.2 Data transformation

The *ResNet18* and *ResNet50* models from *torch.hub* come with specific transformations for input images (see Listing 4.4):

Listing 4.4. Default transformations for ResNet models[2].

```
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])
])
```

Two modifications were made for the training *Supplementary* dataset. First, input resizing was changed, as the original transformation stretched the images, while the new version pads the images with black color to preserve the original aspect ratio (see figure 30). Second, the out-of-the-box normalization values were calculated for the *ImageNet* dataset. To help the model converge faster, the mean and standard deviation were re-calculated for the Supplementary dataset. The final transformation code can be seen in Listing 4.5.

[2] https://pytorch.org/hub/pytorch_vision_resnet/

(a) Stretched

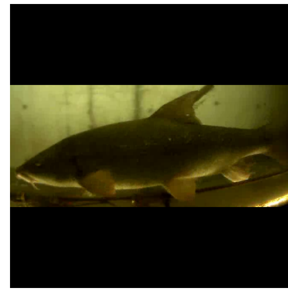

(b) Padded

Figure 30. Visual comparison of a default and a new transformation [40].

Listing 4.5. New transformations for Supplementary training.

```
transform = T.Compose([
    T.Resize(256, max_size=257),
    T.CenterCrop(224),
    T.ToTensor(),
    T.Normalize(mean=[0.2775, 0.2872, 0.2054],
        std=[0.2640, 0.2724, 0.2088]),
])
```

4.2.3 Dataloader and Data sampler

Another thing mentioned in Chapter 3 — class imbalance needed to be addressed. One way to solve it is to either randomly delete images of the major class (undersampling) or randomly duplicate images of the minor class (oversampling) (see Figure 31).

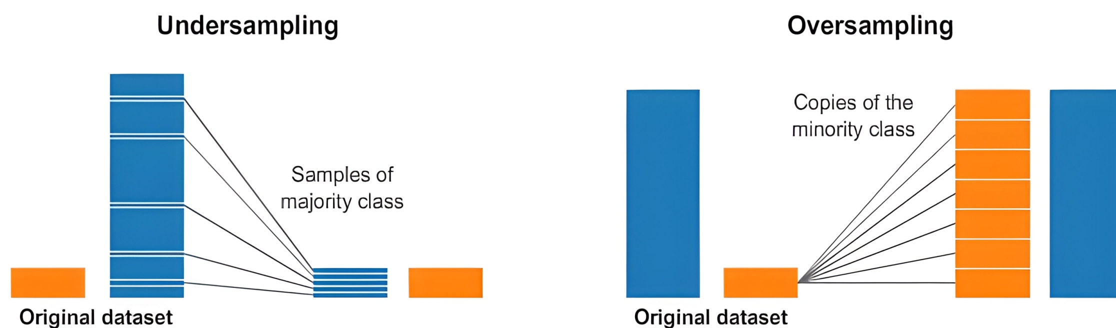


Figure 31. Difference of Undersampling and Oversampling[3].

Here an automatic approach was selected — *torch.utils.data.WeightedRandomSampler*. The main idea is to instead of manually copying and deleting files, *WeightedRandomSampler* picks samples randomly with a gives the probability of images of individual classes.

[3]https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392

To control these probabilities one can use the frequency of representatives (see Listing 4.6).

Listing 4.6. Application of WeightedRandomSampler.

```
_, counts = np.unique(labels, return_counts=True)
weights = 1.0 / torch.tensor(counts, dtype=torch.float)
sampler = WeightedRandomSampler(weights[labels],
    len(labels), replacement=True)
])
```

Then this sampler was plugged-in *torch.utils.data.DataLoader* which distributes images into batches that can be loaded into GPU memory in one go.

4.2.4 Model preparation

After *ResNet18* and *ResNet50* models were retrieved, still they required additional configuration. By default, **ResNet** was designed to solve the **ImageNet** dataset, which has 1000 unique classes. However, in this case, the *Supplementary* dataset consists of only 10 fish species. To accommodate this, the last fully-connected layer of each **ResNet** model was replaced with a custom one (see Listing 4.7):

Listing 4.7. Replacing last fully-connected layer of the ResNet model.

```
model_fc = model.fc
model.fc = nn.Linear(in_features=model_fc.in_features,
    out_features=len(dataset.CLASSES))
])
```

Initially also usage of dropouts [27] was planned, but as it makes the model's convergence slower and *torch* has some problems with saving models with dropout attached, this idea was abandoned.

Besides that also implementations of custom dataset class and model training cycle utilized in this study are available in Appendix 4 and Appendix 5.

4.3 BASELINE model training

Taking into account all prerequisites a baseline model was trained. In particular 5 models (1 model **4vs1** and 4 models **3+1vs1** for each subset) for each model class (*ResNet18* and *ResNet50*). Each *ResNet18* model has completed training in **34 minutes** with an average of **6 minutes** per each subset. *ResNet50* doubles this time to **67 minutes** total and **13 minutes** in average per each subset.

4.4 AutoAugment

As AutoAugment has a very big search space, there are 3 pre-calculated optimal policies for **ImageNet**, **CIFAR**, and **SVHN** datasets that can be utilized for model training on a custom dataset. Unfortunately, no scripts for own policy finding were included within *torch*. There are two possible ways of applying augmentation in torch: either run n times transformation function to increase physically dataset or to generate augmented samples on the fly. The obvious drawback of the first approach is space-consuming, so the second, the simplest, approach was selected.

To apply predefined AA policies the initial transformation was temporarily updated (see an example of applying ImageNet AA policies in Listing 4.8):

Listing 4.8. Plugging AA into transformation function.

```
base_transform = T.Compose([
    T.Resize(256, max_size=257),
    T.CenterCrop(224),
    T.ToTensor(),
    T.Normalize(mean=[0.2775, 0.2872, 0.2054],
        std=[0.2640, 0.2724, 0.2088]),
])
base_transform.transforms.insert(0,
    T.AutoAugment(T.AutoAugmentPolicy.IMAGENET))
```

AA transformation could be plugged directly into *T.Compose* function, but as the training script was put in a cycle, it was a more convenient (and human-error-safe) way of operating.

A total of 30 models were produced ($2 \cdot 5 \cdot 3$), with a total of **37–38 minutes** on *ResNet18* models (per one policy) and **64–65 minutes** on *ResNet50* models.

4.5 Faster AutoAugment

For own policy finding the optimization algorithm proposed as Faster AA was utilized within implementation in *AutoAblument*[4] library.

4.5.1 Optimal policy searching

Before applying *AutoAblument* for searching one has to prepare a dataset class and adjust *search.yaml* config file. A custom dataset class used for all models training (see Appendix 4) was slightly modified: first, as *AutoAblument* is based on *Albumentations* library, the basic transformation function was changed from *torch* implementation to *albumentations* one, luckily the syntax hasn't changed much; second, instead of loading images using *PIL* library this version used *skimage.io* module to load images in a *numpy* format.

Also, the input data were filtered in such a way, that optimal policy searching was done on training data only, to avoid biasing toward test data.

The default configuration used policy searching (modified GAN training) for 20 epochs, this was changed to 300 epochs. Also, like in the case of *ResNet*'s default transformations, the normalization constants were also defined for the ImageNet dataset and were changes to these calculated for *Supplementary* dataset. After searching, optimal policies were saved per each epoch of searching in a *json* file format. A total of **13 hours** were spent on finding optimal policies.

4.5.2 Model training

The same migrating from *torch* transformations to *albumentations* ones were done for model training transformations and found policies were plugged in the following way (see Listing 4.9).

A three policies were chosen: for **20 epochs** (*AutoAblument* defaults), **150 epochs** and **300 epochs**. A total of 30 models were produced ($2 \cdot 5 \cdot 3$), with a total of **33–37 minutes** on *ResNet18* models (per one policy) and **60–63 minutes** on *ResNet50* models.

[4] `https://albumentations.ai/docs/autoalbument/`

Listing 4.9. Found AA policies and updated transformatios[5].

```
transform_pre = A.Sequential([
    A.LongestMaxSize(max_size=256, interpolation=1),
    A.PadIfNeeded(min_height=256, min_width=256, border_mode=0,
        value=(0,0,0)),
    A.CenterCrop(height=224, width=224)
transform = A.load(experiment['policy'])
transform.transforms.insert(0, transform_pre)
])
```

4.6 RandAugment

PyTorch implementation of RA comes with the same hyperparamenters described in original paper [29]: number of sequential operations N and magnitude M. As a starting point the following hyperparameters were used in a grid search to find the best ones (see Table 2).

Table 2. Grid search for the best RandomAugment hyperparameters.

	$\boldsymbol{N = 1}$	$\boldsymbol{N = 2}$	$\boldsymbol{N = 3}$
$\boldsymbol{M = 5}$	RAND_AUGMENT_01	RAND_AUGMENT_07	RAND_AUGMENT_13
$\boldsymbol{M = 7}$	RAND_AUGMENT_02	RAND_AUGMENT_08	RAND_AUGMENT_14
$\boldsymbol{M = 9}$	RAND_AUGMENT_03	RAND_AUGMENT_09	RAND_AUGMENT_15
$\boldsymbol{M = 11}$	RAND_AUGMENT_04	RAND_AUGMENT_10	RAND_AUGMENT_16
$\boldsymbol{M = 13}$	RAND_AUGMENT_05	RAND_AUGMENT_11	RAND_AUGMENT_17
$\boldsymbol{M = 15}$	RAND_AUGMENT_06	RAND_AUGMENT_12	RAND_AUGMENT_18

A total of 180 models were produced $(2 \cdot 5 \cdot 18)$, with a total of **38–43 minutes** on *ResNet18* models (per one type of (N, M)) and **62–69 minutes** on *ResNet50* models. The total time of the grid search was $\approx$ **32 hours**.

4.7 AugMix

The same approach was done for the *PyTorch* implementation of AugMix. The hyperparameters have a similar nature to those in RA. Default parameters in *PyTorch* are **severity** $= \mathbf{3}$ and **mixture_width** $= \mathbf{3}$, so the grid search was done around these values (see Table 3).

[5] *experiment['policy']* contains a path to a *json* file

Table 3. Grid search for the best AugMix hyperparameters.

	width $= 2$	**width** $= 3$	**width** $= 4$
severity $= 1$	AUGMIX_00	AUGMIX_04	AUGMIX_08
severity $= 3$	AUGMIX_01	AUGMIX_05	AUGMIX_09
severity $= 5$	AUGMIX_02	AUGMIX_06	AUGMIX_10
severity $= 7$	AUGMIX_03	AUGMIX_07	AUGMIX_11

A total of 120 models were produced $(2 \cdot 5 \cdot 12)$, with a total of **48–64 minutes** on *ResNet18* models (per one type of $(width, severity)$) and **76–89 minutes** on *ResNet50* models. So far this was the slowest augmentation, with a time increase proportional to increased hyperparameter values. The total time of the grid search was $\approx$ **28 hours**.

4.8 Stable Diffusion: Data preparation

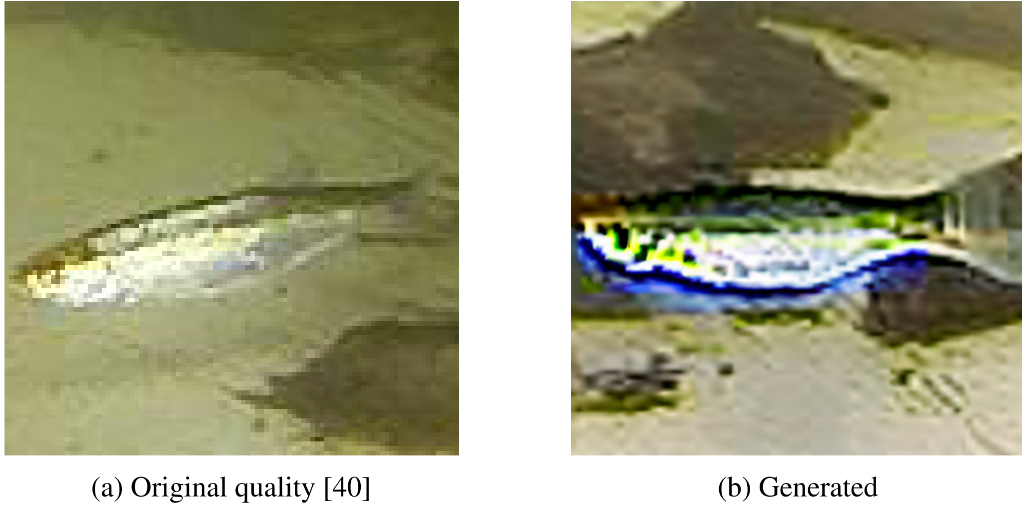
Generative models such as Stable Diffusion differs from the standard augmentation techniques in the way of creating new data — instead of modifying already existing images, it imagines completely new data, which with the side-effect of biasing towards a fine-tuning samples makes it a good candidate for data augmentation.

But before such data can be generated, one has to train (fine-tune) these big generative models. Training such a big model from scratch would require a gigantic amount of training data and computational resources. Instead fine-tuning approaches - *DreamBooth* and *LoRA* are used. A big advantage is that fine-tuning requires much less data, however, the ones used for classification model training are not suitable right away.

In terms of fine-tuning Stable Diffusion train images typically have 512x512 or 768x768 resolution (can be other, but these or most common ones) and are always squared (have a 1:1 aspect ratio). For this work a target resolution was chosen to be 768x768 as it would preserve more small details and, in overall, would increase the quality of generated samples.

4.8.1 Supplementary

A problem with *Supplementary* used for a target model training is that it has various aspect ratios (though in a model transformation, it is padded with black color to fit in a square) and the size of all images is lower than even 512x512. Upscaling it trivially would introduce various image artifacts that will affect generator performance (see Figure 32).

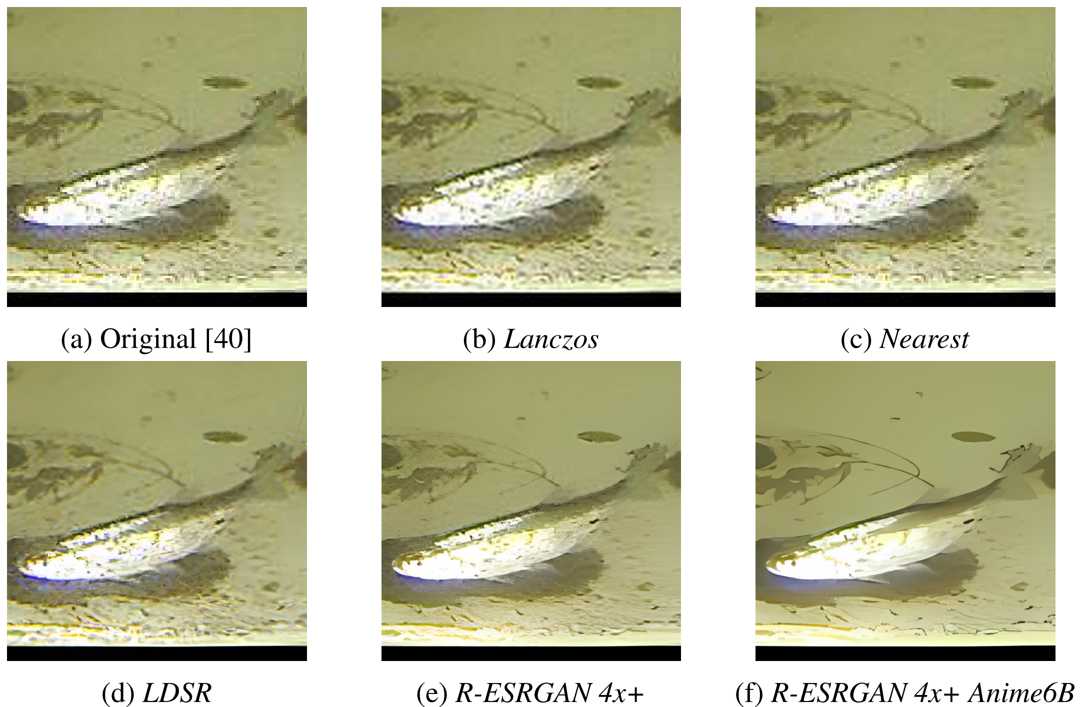


(a) Original quality [40]

(b) Generated

Figure 32. Example of a synthetic *bleak* fish generated from *bad* training samples.

As there was no way of changing data to something else, instead of trivial upscaling, special upscaling models were utilized. It is worth to be mentioned that the analysis and synthetic images generation within Stable Diffusion was done using *stable-diffusion-webui* by Automatic1111[6], which, besides having tools to work with Stable Diffusion, has image upscaling models. Among available ones were: *Lanczos*, *Nearest*, *ESRGAN_4x*, *LDSR*, *R-ESRGAN 4x+*, *R-ESRGAN 4x+ Anime6B* and *SwinIR_4x*. None of the mentioned were producing very decent results, but with *R-ESRGAN 4x+* upscaled image looks less distorted (see Figure 33).



(a) Original [40]

(b) *Lanczos*

(c) *Nearest*

(d) *LDSR*

(e) *R-ESRGAN 4x+*

(f) *R-ESRGAN 4x+ Anime6B*

Figure 33. Example of various upscaling models in *stable-diffusion-webui*.

[6]https://github.com/AUTOMATIC1111/stable-diffusion-webui

Also, to minimize the impact of padding (filling with black color) used for fixing aspect ratios of a target model training, utilizing the same bounding boxes data (see Chapter 3) frames were cropped again in such way that bounding box around the fish were resized to a square using the longest side as a final resolution. Then a total of **150 images** (15 for each fish specie) was selected and upscaled using *R-ESRGAN 4x+* model. The selection was done with a focus on selecting unique images with the best possible quality, which was not uncompromising.

See *Supplementary* dataset for fine-tuning in Appendix 6.

4.8.2 AFFiNe

As was described in Chapter 3, the idea of using *AFFiNe* dataset was to make an additional bias toward the fish domain, which might help Stable Diffusion produce better images in the context of data augmentation. Moreover, besides domain shift, as most of the fish species are present in both *AFFiNe* and *Supplementary*, also the visual appearance of fish present in *AFFiNe* might help distinguish specific species of *Supplementary* where image quality interferes with this.

The situation with *AFFiNe* images was quite the opposite. The resolution of images was much higher than *Supplementary*, but there was no bounding box data and *original frames* to make images squared. There were two possible solutions: either crop the images (fish would also be cropped) or pad them. It was decided that losing the information would have more negative effects rather than leaving black borders, so the second approach was selected. A total of **120 images** were picked (2 species of *Supplementary* are missing in *AFFiNe*) and padded to fit in a 1:1 aspect ratio.

See *AFFiNe* dataset for fine-tuning in Appendix 7).

4.8.3 Images captioning

As was done in the Original paper of *DreamBooth*[20] each species class was mapped with a specific **unique identifier** up to 3 symbols to associate a new knowledge with that identifier, taking manually from pre-calculated file sorted by token frequency of appearance in language[7]. However up-to-date training scripts have an extended functionality by utilizing external caption files. The purpose of that is to describe an image in more detail, which helps the Stable Diffusion to understand the context better.

[7] `https://github.com/2kpr/dreambooth-tokens/blob/main/all_single_tokens_to_4_characters.txt`

To create initial descriptions, a *BLIP* (*model_base_caption_capfilt_large.pth*) model was used, provided with *stable-diffusion-webui*. There were no settings to control the model, but analyzing the code undercover the following configuration (see Listing 4.10):

Listing 4.10. Configuration for BLIP image captioning.

```
{
    ...
    "interrogate_clip_num_beams": 1,
    "interrogate_clip_min_length": 24,
    "interrogate_clip_max_length": 48,
    "interrogate_clip_dict_limit": 1500,
    ...
}
```

Both for *Supplementary* and *AFFiNe* fine-tune datasets captions were created and then manually checked and fixed using a custom *gradio*-based application (see Appendix 3). While for *AFFiNe* automatic captions provided by *BLIP* were okay, describing general information, such as: "*fish is being held in hands*", captions for *Supplementary* were total nonsense, where each image was described as containing various objects and none of them were fish-related. Instead *Supplementary* were manually captioned with a focus on describing visual defects mentioned in Chapter 3[8].

In addition the following prefix structure: "<unique ID> fish, <unique ID> <*Supplementary* label>, <*AFFine* label>" was added to both dataset's captions (see Table 4).

Table 4. Caption prefixes for fine-tuning datasets.

***Supplementary* class**	***AFFiNe* class**	**prefix**
carp	Cyprinus carpio	**bdg** fish, **bdg** barbel, Cyprinus carpio
catfish	Silurus glanis	**cq** fish, **cq** catfish, Silurus glanis
bream	Abramis brama	**ddc** fish, **ddc** bream, Abramis brama
eel	Anguilla anguilla	**fbs** fish, **fbs** eel, Anguilla anguilla
chub	Leuciscus cephalus	**hns** fish, **hns** chub, Leuciscus cephalus
perch	Perca fluviatilis	**pz** fish, **pz** perch, Perca fluviatilis
barbel	Barbus barbus	**rsc** fish, **rsc** barbel, Barbus barbus
roach	Rutilus rutilus	**szn** fish, **szn** roach, Rutilus rutilus
bleak	-	**ccd** fish, **ccd** bleak, Alburnus alburnus
nase	-	**kts** fish, **kts** nase, Chondrostoma nasus

[8]Example prompt: *"... warm white color palette, blurry water, overexposure"*

4.9 Fine-tuning: DreamBooth

The total of 4 models were produced by DreamBooth fine-tuning:

- A: Stable Diffusion v1.5 on *Supplementary* dataset
- B: Stable Diffusion v2.1_768 on *Supplementary* dataset
- C: Stable Diffusion v1.5 on *AFFiNe* dataset
- D: Stable Diffusion v2.1_768 on *AFFiNe* dataset

Each model was trained using the following settings[9] (see Table 5):

Table 5. DreamBooth training settings.

Settings	**Model A**	**Model B**	**Model C**	**Model D**
Base model version	1.5	2.1_768	1.5	2.1_768
U-Net steps	15’000	15’000	15’000	15’000
U-Net lr	2e-6	2e-6	2e-6	2e-6
Encoder steps	6’000	6’000	6’000	6’000
Encoder lr	1e-6	1e-6	1e-6	1e-6
Encoder concept steps	0	0	0	0
Resolution	768	768	768	768
Offset noise	False	False	False	False
External captions	True	True	True	True
Save checkpoint every	3’000	3’000	3’000	3’000

Each model was trained for about **2.5 hours** for text encoder training and about **5.5 hours** for U-Net training (8 hours per model) with a total of **32 hours**. The training was done using *Colab Pro* subscription and **Tesla T4** GPU.

4.10 Fine-tuning: LoRA

Also, the same amount of 4 models was produced by LoRA fine-tuning:

- A: Stable Diffusion v1.5 on *Supplementary* dataset
- B: Stable Diffusion v2.1_768 on *Supplementary* dataset
- C: Stable Diffusion v1.5 on *AFFiNe* dataset
- D: Stable Diffusion v2.1_768 on *AFFiNe* dataset

[9]that were directly visible in a *colab* cell

There is a possibility to install Kohya's script locally or use it in *colab*, but due to major differences in scripts, an older local version was used. The training was using GPU rent service — RunPod[10] using **NVIDIA GeForce RTX 3090** with 24 GB of VRAM and **NVIDIA GeForce RTX 4090** with 24 GB of VRAM GPU's. RunPod utilizes a containers[11] system. To run the training *runpod/stable-diffusion:web-automatic-3.0.0* was used with custom post-installation script (see Appendix 8).

Models based on a 1.5 version were trained using the following command:

Listing 4.11. LoRA training settings for 1.5 version.

```
accelerate launch --num_cpu_threads_per_process=2 "train_network.py" \
--pretrained_model_name_or_path="stabilityai/stable-diffusion-1-5" \
--train_data_dir="/workspace/<dataset_dir>" --resolution=768,768 \
--output_dir="/workspace/<output_dir>" \
--logging_dir="/workspace/<log_dir>" --network_alpha="256" \
--save_model_as=safetensors --network_module=networks.lora \
--text_encoder_lr=5e-5 --unet_lr=0.0001 --network_dim=256 \
--output_name="<model_name>" --lr_scheduler_num_cycles="300" \
--learning_rate="0.0001" --lr_scheduler="cosine" \
--lr_warmup_steps="<900 or 1125>" --train_batch_size="4" \
--max_train_steps="< 9000 or 11250>" --save_every_n_epochs="100" \
--mixed_precision="bf16" --save_precision="bf16" --seed="1234" \
--caption_extension=".txt" --cache_latents --optimizer_type="AdamW8bit" \
--max_token_length=225 --bucket_reso_steps=64 --flip_aug --xformers --bucket_no_upscale
```

Models based on a 2.1 version were trained using the following command:

Listing 4.12. LoRA training settings for 2.1 version.

```
accelerate launch --num_cpu_threads_per_process=2 "train_network.py" \
--v2 --v_parameterization \
--pretrained_model_name_or_path="stabilityai/stable-diffusion-2-1" \
... rest is the same for 1.5 version
```

Each model was fully trained **3.5 hours** on RTX 3090 and **2 hours** on RTX 4090, which is much faster taking into account that using *LoRA* both text encoder and U-Net were trained 3 times longer (according to the number of iteration steps) while taking 2–3 times less time than *DreamBooth*. The whole training was done in about **12 hours**.

[10] https://www.runpod.io/
[11] https://www.docker.com/

4.11 Checkpoint selection

After 8 fine-tuned models were trained, there still were multiple versions to select for further synthetic dataset generation. Regarding *DreamBooth* each model was saved every 3000 steps, but only one of each was chosen. As for *LoRA*, despite having the same saving each n epoch, only the final versions were used, however as *LoRA* have mixing constant α, the selection of optimal one was analyzed.

Dealing with the combination of *Supplementary* and *AFFiNe* was different. The initial plan was to use fine-tuned with *AFFiNe* data model and fine-tune it further with *Supplementary* data. However, trying that using *DreamBooth* results in an error both on 1.5 and 2.1-based versions. Also, merging a *LoRA* with a base model[12] was not working for 2.1 based model. To solve these problems an alternative approach was selected — merge different models together[13].

Checkpoint comparison plots that were used to decide on a final selection can be seen in Appendix 9. The general rule was to avoid visual defects and at the same time avoid high biasing toward the original data (overfitting). It is worth to be mentioned that none of the models were perfect, some have broken fish, and some have not a fish at all[14] See the final models used for synthetic data generation in Table 6:

Table 6. Synthetic data generation final candidates.

Name	**Recipe**
sup_v1.5_lora	v1.5 base + 0.8 · sup_768_v1.5
sup_v2.1_lora	v2.1 base + 0.8 · sup_768_v2.1
sup_v1.5_dream	sup_768_1.5_10_step_15000
sup_v2.1_dream	sup_768_2.1_10_step_15000
sup+_v1.5_lora	sup_v1.5_lora + 0.5 · affine_768_v1.5
sup+_v2.1_lora	sup_v2.1_lora + 0.5 · affine_768_v2.1
sup+_v1.5_dream	0.6 · sup_v1.5_dream + 0.4 · affine_768_1.5_10_step_12000
sup+_v2.1_dream	0.5 · sup_v2.1_dream + 0.5 · affine_768_2.1_10_step_15000

[12]not the same as applying a *LoRA* with a base model
[13]that have the same base model and fine-tune approach
[14]for example *szn roach* was interpreted by some models as *cockroach*

4.12 Synthetic datasets generation

As there is no obvious answer on the optimal amount of synthetic data, using each model, **1000 synthetic images** (100 images per specie) were generated, with a total of **8000 synthetic images**. Each set generation took approximately **3.5–4 hours** with a total of around **30 hours**. Examples of generated datasets are presented in Appendix 10.

4.13 Model training using synthetic datasets

Using created synthetic datasets a pair of *ResNet18* and *ResNet50* were trained. No additional transformations were necessary, just original and synthetic data were combined for each set of synthetic data. A synthetic set was labeled as *SD* and mixed into each of the 5 training subsets.

A total of 80 models were produced ($2 \cdot 5 \cdot 8$), with a total of **43–45 minutes** on *ResNet18* models and **88–94 minutes** on *ResNet50* models. Such an increase is not surprising taking into account the increase of the training samples and hence the increase of iteration steps.

In addition, a pair of models were trained on synthetic subsets combined together. The resulting dataset increased from 2k to 10k samples. Resulting training time increase to **2.5 hours** and **4.5 hours** respectively.

4.14 Model training using a mix of augmentations

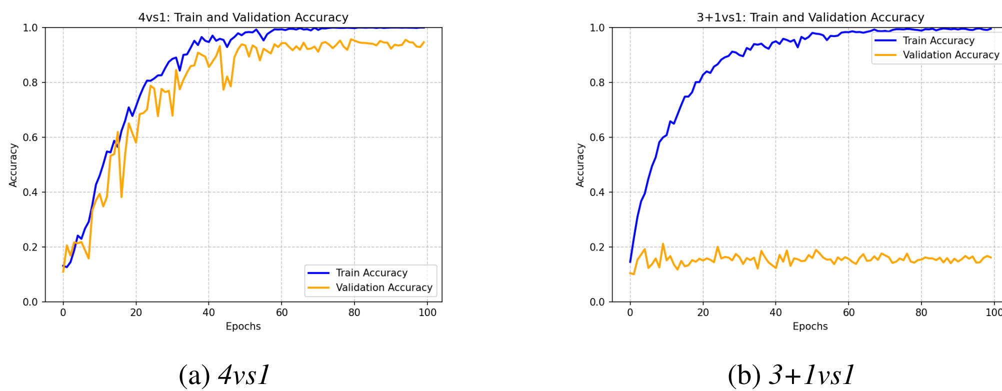
Lastly, a combination of all synthetic data together and RA ($N = 2, M = 9$) were trained the same way. As RA did not increase time much previously, resulting training time remains approximately the same, **2.5 hours** and **4.5 hours** respectively.

5. Validation and results

In this chapter, the results of the experiments are presented, including baseline model evaluation, the impact of different augmentation techniques target model, evaluation of various metrics and total time spent on each technique as well as the selection of the best approach and unresolved problems discussion.

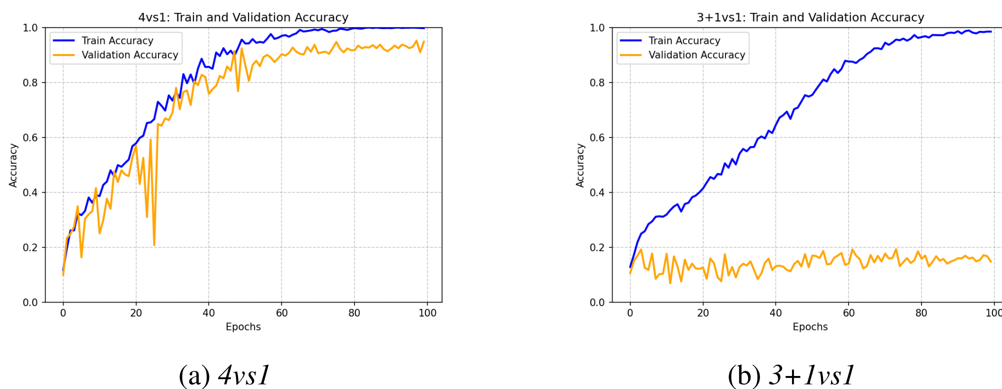
5.1 BASELINE model evaluation

To understand the effect of augmentation and select potentially the best one, it is necessary to understand the baseline performance. The first thing is as was mentioned in Chapters 2 and 4, there is indeed a clear presence of data leakage that can be seen in Figures 34–35:



(a) *4vs1* (b) *3+1vs1*

Figure 34. Baseline *ResNet18* model training and validation accuracy.

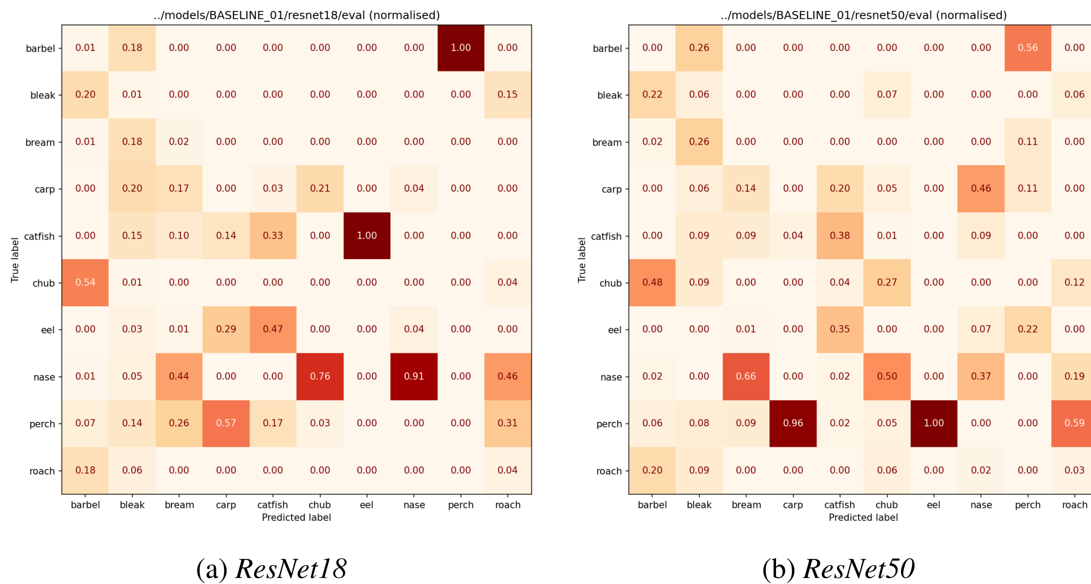


(a) *4vs1* (b) *3+1vs1*

Figure 35. Baseline *ResNet50* model training and validation accuracy.

Models that were trained on *4vs1* subset were both trained and validation set to share the saved video origins. While validation accuracy is pretty good, the reality of *3+1vs1* shows a clear case of overfitting. This effect stays present for both *ResNet18* and *ResNet50* architectures, though *ResNet50* shows a more linear increase in training accuracy.

The confusion matrices of baseline models are available in Figure 36. To properly display class imbalance the values of each confusion matrix were normalized over the prediction axis:



(a) *ResNet18* (b) *ResNet50*

Figure 36. Normalized confusion matrices of baseline models.

As can be seen, there are not many values over the main diagonal in both matrices, which should be evidence that the model is not able to predict anything. While *ResNet18* was able to predict one class almost correctly — *nase*, both of the models show more random guessing behavior. It should be noted here, that the accuracy of label prediction of 10 classes is in the range of 0.1 up to 1, where 0.1 is random picking. Table 7 shows that unfortunately baseline models indeed predict the label not better than random guessing.

Table 7. Baseline models metrics.

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	**F_1-score**
ResNet18	0.070	0.133	0.065	0.899	0.068
ResNet50	0.130	0.111	0.106	0.904	0.101

5.2 Model training with augmentation evaluation

Then the analysis and the selection of the best model was done. The selection is on the basis of the metrics, described in Chapter 2. It should be noted that, unlike other domains, there were no preferences toward any specific metric.

5.2.1 Selection of RandAugment

The tables below present the evaluation metrics for *ResNet18* and *ResNet50* with the application of *RandAugment* technique. The hyperparameters behind each experiment are available in the corresponding section of Chapter 4. One immediate notice is that in overall evaluation metrics of *ResNet18* and *ResNet50* were proportional to the hyperparameter's values increase. Among *ResNet18* models the **RAND_AUGMENT_16** was selected as it has both top *recall* and *F1-score*, as well as decent *accuracy* and *precision*.

Table 8. RandAugment augmentation metrics on *ResNet18* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
RAND_AUGMENT_01	0.256	0.356	0.186	0.918	0.202
RAND_AUGMENT_02	0.256	0.177	0.190	0.917	0.171
RAND_AUGMENT_03	0.252	0.152	0.163	0.916	0.150
RAND_AUGMENT_04	0.222	0.143	0.144	0.914	0.133
RAND_AUGMENT_05	0.258	0.211	0.182	0.918	0.175
RAND_AUGMENT_06	0.240	0.273	0.163	0.915	0.164
RAND_AUGMENT_07	0.250	0.355	0.182	0.918	0.209
RAND_AUGMENT_08	0.303	0.366	0.230	0.921	0.225
RAND_AUGMENT_09	0.232	0.291	0.158	0.914	0.162
RAND_AUGMENT_10	0.295	0.354	0.229	0.923	0.234
RAND_AUGMENT_11	0.272	0.330	0.194	0.918	0.206
RAND_AUGMENT_12	0.301	0.209	0.200	0.922	0.201
RAND_AUGMENT_13	0.326	**0.426**	**0.225**	0.925	0.239
RAND_AUGMENT_14	0.297	0.305	0.205	0.921	0.230
RAND_AUGMENT_15	0.286	0.271	0.194	0.920	0.205
RAND_AUGMENT_16	0.324	0.397	**0.255**	**0.926**	**0.303**
RAND_AUGMENT_17	0.292	0.305	0.220	0.920	0.240
RAND_AUGMENT_18	**0.330**	0.289	0.231	**0.926**	0.246

The same way **RAND_AUGMENT_18** was selected among *ResNet50* models, though there was also *RAND_AUGMENT_15* as a decent option:

Table 9. RandAugment augmentation metrics on *ResNet50* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
RAND_AUGMENT_01	0.258	0.290	0.198	0.917	0.199

Continues...

Table 9 – *Continues...*

Model	Accuracy	Precision	Recall	TNR	F_1-score
RAND_AUGMENT_02	0.232	0.223	0.151	0.915	0.144
RAND_AUGMENT_03	0.279	0.164	0.176	0.920	0.166
RAND_AUGMENT_04	0.240	0.173	0.177	0.915	0.158
RAND_AUGMENT_05	0.274	0.242	0.192	0.918	0.189
RAND_AUGMENT_06	0.258	0.217	0.170	0.918	0.181
RAND_AUGMENT_07	0.277	0.348	0.195	0.919	0.207
RAND_AUGMENT_08	0.272	0.322	0.196	0.918	0.214
RAND_AUGMENT_09	0.295	0.341	0.206	0.921	0.210
RAND_AUGMENT_10	0.267	0.296	0.193	0.918	0.213
RAND_AUGMENT_11	0.225	0.342	0.162	0.913	0.183
RAND_AUGMENT_12	0.288	0.317	0.172	0.920	0.186
RAND_AUGMENT_13	0.258	0.338	0.186	0.917	0.200
RAND_AUGMENT_14	0.277	**0.389**	0.205	0.919	0.231
RAND_AUGMENT_15	0.306	0.328	**0.228**	0.923	**0.247**
RAND_AUGMENT_16	0.263	0.285	0.193	0.918	0.205
RAND_AUGMENT_17	0.227	0.268	0.159	0.915	0.155
RAND_AUGMENT_18	**0.337**	0.326	**0.227**	**0.926**	0.233

5.2.2 Selection of AugMix

The situation with the application of *AugMix* was quite the opposite to *RandAugment* as the experiments with lower hyperparameter's values have achieved better results (see Tables 10–11). Again all hyperparameters behind each experiment are available in the corresponding section of Chapter 4. Among *ResNet18* models **AUGMIX_01** was selected.

Table 10. AugMix augmentation metrics on *ResNet18* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
AUGMIX_00	0.265	0.320	0.172	0.920	0.180
AUGMIX_01	**0.344**	**0.397**	0.245	**0.928**	0.228
AUGMIX_02	0.321	0.308	0.239	0.926	0.213
AUGMIX_03	0.303	0.250	0.225	0.924	0.206
AUGMIX_04	0.247	0.244	0.155	0.918	0.175
AUGMIX_05	0.330	0.266	0.238	0.927	0.213
AUGMIX_06	0.328	0.309	0.243	0.926	**0.231**

Continues...

Table 10 – *Continues...*

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	**F_1-score**
AUGMIX_07	0.256	0.229	0.197	0.919	0.170
AUGMIX_08	0.279	0.229	0.197	0.921	0.187
AUGMIX_09	0.333	0.244	**0.246**	0.927	0.217
AUGMIX_10	0.306	0.361	0.220	0.925	0.204
AUGMIX_11	0.277	0.210	0.203	0.921	0.180

As for *ResNet50* models — **AUGMIX_02** showed the best evaluation results.

Table 11. AugMix augmentation metrics on *ResNet50* model.

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	**F_1-score**
AUGMIX_00	0.263	0.330	0.203	0.918	0.214
AUGMIX_01	0.301	**0.393**	0.223	0.924	0.203
AUGMIX_02	**0.341**	0.345	0.241	**0.928**	0.226
AUGMIX_03	0.259	0.218	0.192	0.918	0.176
AUGMIX_04	0.290	0.325	0.204	0.922	0.211
AUGMIX_05	0.315	0.307	0.227	0.925	0.208
AUGMIX_06	0.323	0.314	0.236	0.926	0.221
AUGMIX_07	0.333	0.267	0.245	0.927	0.220
AUGMIX_08	0.292	0.293	0.208	0.923	0.214
AUGMIX_09	0.330	0.333	**0.248**	0.927	**0.235**
AUGMIX_10	0.276	0.263	0.197	0.921	0.184
AUGMIX_11	0.249	0.348	0.188	0.918	0.171

5.2.3 Faster AutoAugment search

Faster AA requires additional policy searching, evaluation of which in *AutoAlbument* library is done on the performance of the small *cnn*, which in reality was *ResNet18*. See the training loss in Figure 37. While there was a clear loss drop which suggests that there was a positive effect of the found policies, the application of them on target models was not very successful. The numbers near each experiment refer to the search epochs elapsed for each selected policy: **20 epochs**, **150 epochs**, and **300 epochs**.

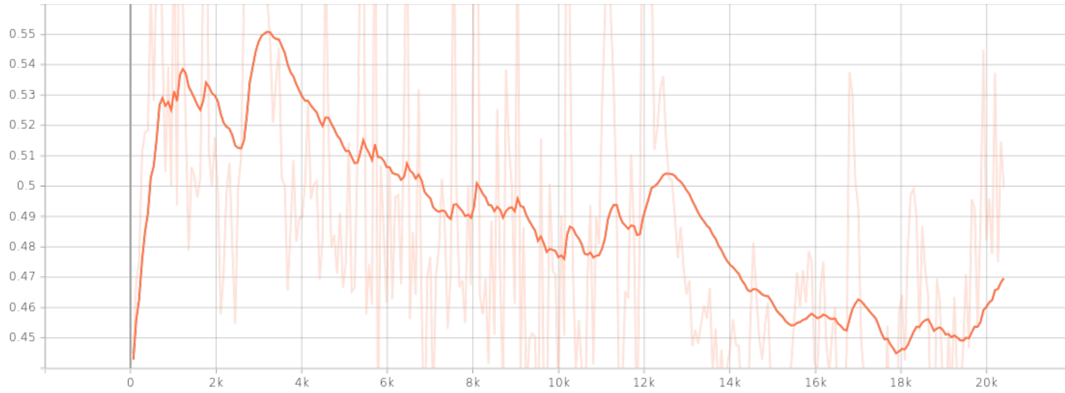


Figure 37. Faster AA search training loss.

Surprisingly the *ResNet18* models performances (see Table 12) were in a linear relationship to the epoch number of selected policies, with 300 epochs achieving the best results, probably due to the same *ResNet18* model architecture used both for policy searching and evaluation. However, the results are still not very interesting as it was close to random guessing. It might be that with even further training this approach might produce better results, but as the policy searching itself consumes a lot of time, this idea was not investigated further.

Table 12. Faster AA augmentation metrics on *ResNet18* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
FASTER_AA_01	0.119	0.088	**0.108**	0.901	0.091
FASTER_AA_02	0.110	0.102	0.092	0.901	0.089
FASTER_AA_03	**0.121**	**0.113**	**0.108**	**0.903**	**0.098**

With *ResNet50* models (see Table 13), there was no such effect, and the results were not particularly interesting, as other augmentation techniques achieved better evaluation metrics.

Table 13. Faster AA augmentation metrics on *ResNet50* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
FASTER_AA_01	0.160	0.136	**0.149**	0.907	**0.133**
FASTER_AA_02	**0.205**	**0.143**	0.134	**0.913**	**0.133**
FASTER_AA_03	0.142	0.119	0.090	0.906	0.101

5.2.4 Synthetic datasets

With the use of synthetic datasets, 3 approaches, in general, were evaluated:

- One synthetic dataset per experiment
- All synthetic datasets together in one experiment
- As the previous one but with extra *RandAugment* application

Among the first approach dataset generated using Stable Diffusion v2.1 fine-tuned by *LoRA* show decent results both in the case of *ResNet18* and *ResNet50* (see Tables 14–15). The combined approach has increased the metrics even further, making it comparable with *RandAugment* and *AugMix* augmentation techniques. But the absolute best results were achieved using the almost-accidental approach of combining both the synthetic data and standard augmentation techniques like *RandAugment*.

Table 14. Training with synthetic data on *ResNet18* model.

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	F_1**-score**
SD_SUP_LORA_15	0.180	0.206	0.114	0.910	0.130
SD_SUP_LORA_21	**0.259**	**0.254**	**0.165**	**0.919**	**0.179**
SD_SUP_DREAM_15	0.196	0.218	0.151	0.913	0.163
SD_SUP_DREAM_21	0.142	0.128	0.093	0.907	0.102
SD_SUP+_LORA_15	0.173	0.170	0.119	0.910	0.130
SD_SUP+_LORA_21	0.207	0.144	0.124	0.913	0.129
SD_SUP+_DREAM_15	0.119	0.110	0.070	0.904	0.083
SD_SUP+_DREAM_21	0.168	0.160	0.119	0.909	0.128
SD_SUP_COMBINED	**0.346**	**0.342**	**0.252**	**0.928**	**0.239**
SD_SUP_COMBINED_RAND	**0.575**	**0.503**	**0.440**	**0.952**	**0.418**

Table 15. Training with synthetic data on *ResNet50* model.

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	F_1**-score**
SD_SUP_LORA_15	0.184	0.186	0.127	0.911	0.136
SD_SUP_LORA_21	**0.321**	**0.202**	**0.240**	**0.925**	**0.204**
SD_SUP_DREAM_15	0.186	0.137	0.124	0.912	0.130
SD_SUP_DREAM_21	0.195	0.166	0.124	0.911	0.126
SD_SUP+_LORA_15	0.196	0.180	0.128	0.913	0.138

Continues...

Table 15 – *Continues...*

Model	Accuracy	Precision	Recall	TNR	F_1-score
SD_SUP+_LORA_21	0.142	0.157	0.094	0.906	0.101
SD_SUP+_DREAM_15	0.169	0.147	0.092	0.908	0.098
SD_SUP+_DREAM_21	0.178	0.151	0.113	0.910	0.123
SD_SUP_COMBINED	**0.328**	**0.253**	0.238	**0.926**	**0.235**
SD_SUP_COMBINED_RAND	**0.510**	**0.408**	**0.387**	**0.944**	**0.359**

5.2.5 Total metrics comparison

Then each best result was further compared among each other (see Tables 16–17). While both *RandAugment* and *AugMix* show decent results and for some metrics that was the case for *AutoAugment* pre-defined policy for *ImageNet* dataset, the best results were achieved using a combination of synthetic data and *RandAugment*, beating every other technique by every evaluation metric both for *ResNet18* and *ResNet50* models.

Table 16. Total metrics comparison on *ResNet18* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
BASELINE	0.070	0.133	0.065	0.899	0.068
AA_CIFAR10	0.314	0.359	0.240	0.923	0.253
AA_IMAGENET	0.348	0.328	0.270	0.927	0.268
AA_SVHN	0.252	0.284	0.162	0.918	0.195
RAND_AUGMENT_16	0.324	**0.397**	0.255	0.926	0.303
AUGMIX_01	0.344	**0.397**	0.245	0.928	0.228
FASTER_AA_03	0.121	0.113	0.108	0.903	0.098
SD_SUP_LORA_21	0.259	0.254	0.165	0.919	0.179
SD_SUP_COMBINED	0.346	0.342	0.252	0.928	0.239
SD_SUP_COMBINED_RAND	**0.575**	**0.503**	**0.440**	**0.952**	**0.418**

Table 17. Total metrics comparison on *ResNet50* model.

Model	Accuracy	Precision	Recall	TNR	F_1-score
BASELINE	0.130	0.111	0.106	0.904	0.101
AA_CIFAR10	0.265	0.297	0.199	0.918	0.200
AA_IMAGENET	0.312	0.344	0.256	0.923	0.253

Continues...

Table 17 – *Continues...*

Model	Accuracy	Precision	Recall	TNR	F_1-score
AA_SVHN	0.234	0.179	0.151	0.915	0.147
RAND_AUGMENT_18	0.337	0.326	0.227	0.926	0.233
AUGMIX_02	0.341	0.345	0.241	0.928	0.226
FASTER_AA_02	0.205	0.143	0.134	0.913	0.133
SD_SUP_LORA_21	0.321	0.202	0.240	0.925	0.204
SD_SUP_COMBINED	0.328	0.253	0.238	0.926	0.235
SD_SUP_COMBINED_RAND	**0.510**	**0.408**	**0.387**	**0.944**	**0.359**

5.2.6 Total spent time comparison

But before the final evaluation of the best models so far, a few words should be addressed regarding the time spent on preparing and application of each augmentation technique (see Table 18):

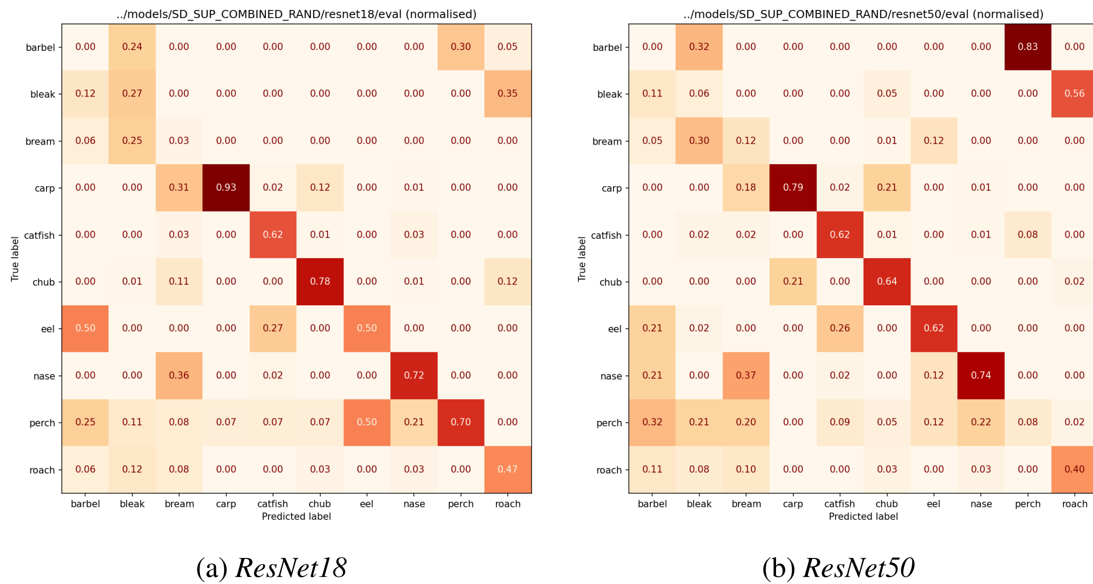
Table 18. Total time spent on each technique (*hours:minutes*).

Approach	ResNet18	ResNet50	Preparation	Total
BASELINE	00:34	01:07	00:00	00:34–01:07
AutoAugment (grid search)	00:38	01:01	00:00	00:38–01:01
RandAugment (grid search)	00:40	01:06	12:00 + 20:00	12:40–21:06
AugMix	00:56	01:22	11:12 + 16:30	12:08–17:52
Faster AA	00:35	00:62	13:00	13:35–13:62
Synthetic data	00:44	01:30	(03:00–08:00) + 04:00	10:14–11:00
Combined synthetic data	02:30	04:30	12:00 + 32:00 + 30:00	76:30–78:30
Combined synthetic data + RandAugment	02:30	04:30	12:00 + 32:00 + 30:00	76:30–78:30

As can be seen, despite a combinational approach of synthetic data together with *RandAugment* having the best result with regards to evaluation metrics, it is far from the most time-efficient approach. The standard popular approaches like *RandAugment* and *AugMix* have much less time as if one commits the grid search, they are the fastest way, especially to test proof-of-concept. If one includes grid search time, there is a more time-efficient approach —utilizing Stable Diffusion generation on a single model. Though it will not produce the best results always, in addition to synthetic data for model training, there is a useful artifact of fine-tuned Stable Diffusion model that can be utilized again and again and with proper evaluation can be used in similar domains.

5.3 Best models evaluation

So far the best model was achieved using a combination of synthetic data and *RandAugment*. The same normalized confusion matrices are presented for the best models in Figure 38. Unline the baseline model by looking at the main diagonals there are certain improvements in distinguishing between classes. However, there are still some problems, such as predicting *perch* as *barbel*.



(a) *ResNet18* (b) *ResNet50*

Figure 38. Normalized confusion matrices of *SD_SUP_COMBINED_RAND* models.

Overall, in comparison with the baseline model, the proposed approach increased *accuracy* and *precision* metrics by 5 times and the rest by 3.5-4 times (except for *TNR* where values were already close to the 1) (see Tables 19–20).

Table 19. Baseline models metrics.

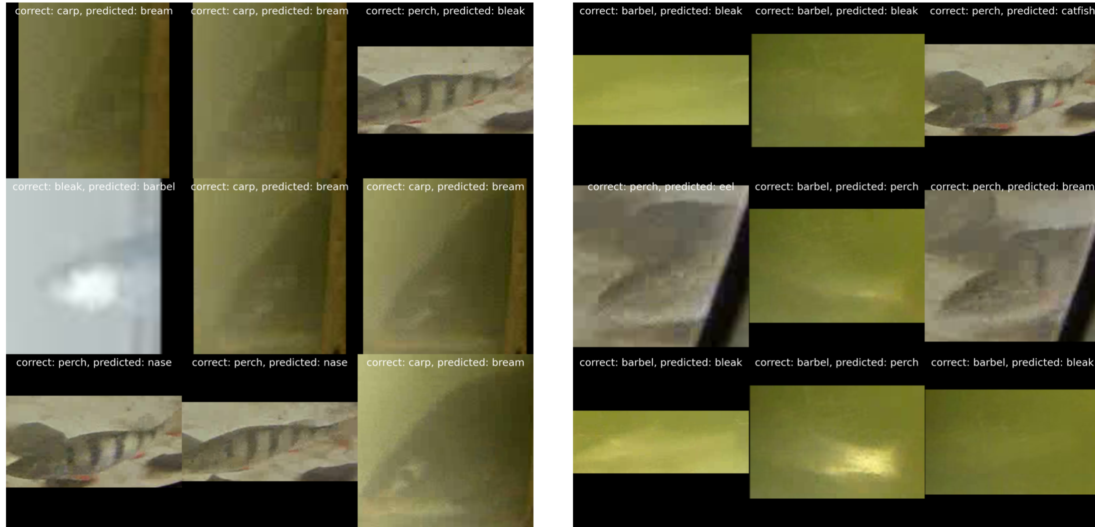
Model	**Accuracy**	**Precision**	**Recall**	**TNR**	F_1**-score**
ResNet18	0.070	0.133	0.065	0.899	0.068
ResNet50	0.130	0.111	0.106	0.904	0.101

Table 20. Best achieved models metrics.

Model	**Accuracy**	**Precision**	**Recall**	**TNR**	F_1**-score**
ResNet18	0.575	0.503	0.440	0.952	0.418
ResNet50	0.510	0.408	0.387	0.944	0.359

5.4 Problem analysis

As the achieved results are still far from being ideal[1] an additional analysis was applied. Using CleanLab[2] predicted labels were analyzed to determine the largest label error[3]. Examples of images with the largest label errors can be seen in Figure 39:



(a) *SD_SUP_COMBINED_RAND ResNet18* (b) *SD_SUP_COMBINED_RAND ResNet50*

Figure 39. Examples of images with an incorrectly predicted label [40].

As can be seen, the majority of images with incorrect labels have very tough visual conditions and to be more specific, probably even a human will not be able to correctly classify most of them. The interesting was the prediction of *perch* fish, as it has quite a unique appearance among other species that were present in the *Supplementary* dataset. However, as CNN models have no common-sense understanding (typically), most probably the stones behind the fish were interpreted as part of that fish, which, from the perspective of models, makes fish larger, and hence the choice was made in favor of larger fish species, like *catfish*.

In addition, comparing different models on this test dataset might have introduced extra challenges, however, this choice was made to maintain the original quality of the data.

[1] close to 1 (or 100%)
[2] `https://github.com/cleanlab/cleanlab`
[3] in the case where the predicted label was not the same as true one

6. Discussion and further work

Throughout the study, a number of important questions were addressed, leading to key findings and insights. The optimal combination of augmentation techniques for improving the performance of a specific model was discovered to be synthetic images generated using Stable Diffusion fine-tuned with *Supplementary* and *AFFiNe* datasets, combined with additional *RandAugment* transformations. This combination allowed for better generalization of positional invariances of fish in the images, which lead to the increase of evaluation metrics by 4-5 times.

Additionally, it was determined that it is possible to evaluate the effect of augmentation techniques in the context of low-quality limited-data dataset scenarios, where due to some factors, alternative ways of collecting more real data are not financially profitable or time-consuming. However, this process turned out to be quite challenging, and the improvements observed were modest. Despite this, the results demonstrated the potential for novel augmentation techniques, such as diffusion models, to be worth applying in low-quality limited-data scenarios, compared to traditional ones. Although considerable time was spent on fine-tuning and further synthetic data generation, other techniques did not lead to performance improvements that outperform those achieved with Stable Diffusion alone and even matched those achieved with a mixed approach. Moreover, the produced Stable Diffusion models have potential applications beyond synthetic data generation for model training, as they are capable of recreating trained objects in different contexts.

Both the results and described workflow provide a wide roadmap for further research. A further step could be to prune the synthetic data before using it as training data: for example, make 100000 synthetic images, and pick only those which are as similar to "real images" as possible, based on some threshold. Researchers might also continue investigating and improving the proposed solution, perhaps by creating better prompts, training a custom image upscaling model to fine-tune Stable Diffusion with low-quality images of fish, or exploring alternative augmentation techniques that could yield better results. Another, more traditional-popular, possible avenue for future research would be to focus on collecting more real data to enhance the training dataset and then using that together with the proposed solution to achieve even better results.

7. Summary

In conclusion, this study addressed the challenges in fish species classification in the context of data-limited, low-quality datasets. The study has investigated and evaluated various image augmentation techniques, aiming to improve the performance of *ResNet18* and *ResNet50* models for fish species classification.

The results demonstrated that a combination of synthetic images generated using Stable Diffusion fine-tuned with *Supplementary* and *AFFiNe* datasets, along with additional *RandAugment* transformations, yielded the best performance improvements in the specific model, increasing the evaluation metrics by 4-5 times. The study also highlighted the potential for novel augmentation techniques, such as diffusion models, in particular fine-tuned Stable Diffusion, to be worth applying compared to traditional ones. Moreover, the study explored the practicability of applying advanced image augmentation techniques to small initial datasets and assessed their effectiveness in mitigating issues related to dataset imbalances and overfitting. While the improvements observed were modest due to the limited, low-quality data available, the study showed that advanced image augmentation techniques can be effectively applied in limited data scenarios.

The results of this work have several implications for the field of fish species classification and the broader application of image augmentation techniques. By identifying optimal combinations of augmentation techniques and demonstrating the potential of novel methods, the study contributes to the advancement of fish species classification models, which can be used to monitor and preserve underwater ecosystems and support various economic activities.

Future work in this area could focus on further investigation of the proposed solution by improving the prompts used, training custom image upscaling models, or investigating alternative augmentation techniques. Additionally, researchers could concentrate on collecting more real data to enhance the training dataset, further improving the performance of fish species classification models. The potential applications of the Stable Diffusion models beyond synthetic data generation for model training could also be explored, contributing to the broader understanding of these techniques.

The source code of all the experiments is available at *GitLab*.

8. Acknowledgements

The author would like to thank the German Federal Institute of Hydrology (BfG) for allowing the use of the *Supplementary* dataset. The *Supplementary* dataset was provided by Bundesanstalt für Gewässerkunde / Federal Institute of Hydrology as a part of "**BfG-Project Smart fish counter for monitoring species, size, migration behavior and environmental conditions**" project[1].

The opportunity to work on this project as part of a collaboration between research groups at TalTech and BfG has been greatly appreciated. The guidance, experience, and cooperation of both parties played an important role in the workflow and results of this study. The author is grateful for the continued support, encouragement, and general enthusiasm for the subject shown by all project participants.

[1]https://www.etis.ee/Portal/Projects/Display/669139e4-524b-4562-9aad-a1164b870823

References

[1] Jürgen Soom et al. "Environmentally adaptive fish or no-fish classification for river video fish counters using high-performance desktop and embedded hardware". In: *Ecological Informatics* 72 (2022). [Accessed: 22-04-2023], p. 101817. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2022.101817. URL: https://www.sciencedirect.com/science/article/pii/S1574954122002679.

[2] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989). [Accessed: 19-03-2023], pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[3] Dhruv Rathi, Sushant Jain, and Dr. S. Indu. *Underwater Fish Species Classification using Convolutional Neural Network and Deep Learning*. [Accessed: 21-03-2023]. 2018. arXiv: 1805.10106 [cs.CV].

[4] Mutasem K. Alsmadi and Ibrahim Almarashdeh. "A survey on fish classification techniques". In: *Journal of King Saud University - Computer and Information Sciences* 34.5 (2022). [Accessed: 21-03-2023], pp. 1625–1638. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2020.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1319157820304195.

[5] Alexander Buslaev et al. "Albumentations: Fast and Flexible Image Augmentations". In: *Information* 11.2 (Feb. 2020). [Accessed: 19-03-2023], p. 125. DOI: 10.3390/info11020125. URL: https://doi.org/10.3390%5C%2Finfo11020125.

[6] Mingle Xu et al. *A Comprehensive Survey of Image Augmentation Techniques for Deep Learning*. [Accessed: 21-03-2023]. 2022. arXiv: 2205.01491 [cs.CV].

[7] Teerath Kumar et al. *Image Data Augmentation Approaches: A Comprehensive Survey and Future directions*. [Accessed: 21-03-2023]. 2023. arXiv: 2301.02830 [cs.CV].

[8] Marcus D. Bloice, Christof Stocker, and Andreas Holzinger. *Augmentor: An Image Augmentation Library for Machine Learning*. [Accessed: 19-03-2023]. 2017. arXiv: 1708.04680 [cs.CV].

[9] Ian J. Goodfellow et al. *Generative Adversarial Networks*. [Accessed: 19-03-2023]. 2014. arXiv: 1406.2661 [stat.ML].

[10] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. [Accessed: 19-03-2023]. 2022. arXiv: 1312.6114 [stat.ML].

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. [Accessed: 19-03-2023]. 2020. arXiv: 2006.11239 [cs.LG].

[12] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. [Accessed: 19-03-2023]. 2022. arXiv: 2112.10752 [cs.CV].

[13] Tara M. Pattilachan et al. *A Critical Appraisal of Data Augmentation Methods for Imaging-Based Medical Diagnosis Applications*. [Accessed: 19-03-2023]. 2022. arXiv: 2301.02181 [eess.IV].

[14] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. [Accessed: 19-03-2023]. 2021. arXiv: 2102.09672 [cs.LG].

[15] Alex Nichol et al. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. [Accessed: 19-03-2023]. 2022. arXiv: 2112.10741 [cs.CV].

[16] Suorong Yang et al. *Image Data Augmentation for Deep Learning: A Survey*. [Accessed: 21-03-2023]. 2022. arXiv: 2204.08610 [cs.CV].

[17] Ziqiang Zheng et al. "Fish Recognition from a Vessel Camera Using Deep Convolutional Neural Network and Data Augmentation". In: *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*. [Accessed: 21-03-2023]. May 2018, pp. 1–5. DOI: 10.1109/OCEANSKOBE.2018.8559314.

[18] Abdelouahid Ben Tamou, Abdesslam Benzinou, and Kamal Nasreddine. "Targeted Data Augmentation and Hierarchical Classification with Deep Learning for Fish Species Identification in Underwater Images". In: *Journal of Imaging* 8.8 (2022). [Accessed: 21-03-2023]. ISSN: 2313-433X. DOI: 10.3390/jimaging8080214. URL: https://www.mdpi.com/2313-433X/8/8/214.

[19] Erik Dzotsenidze. *Generative Adversarial Networks as a Data Augmentation Tool for CNN-based Parkinson's Disease Diagnostics*. [Accessed: 21-03-2023]. URL: https://digikogu.taltech.ee/en/Item/101ba2d0-0229-4a38-96fa-2671988503ba.

[20] Nataniel Ruiz et al. *DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation*. [Accessed: 19-03-2023]. 2023. arXiv: 2208.12242 [cs.CV].

[21] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. [Accessed: 19-03-2023]. 2021. arXiv: 2106.09685 [cs.CL].

[22] Brandon Trabucco et al. *Effective Data Augmentation With Diffusion Models*. [Accessed: 19-03-2023]. 2023. arXiv: 2302.07944 [cs.CV].

[23] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. [Accessed: 21-03-2023]. 2021. arXiv: 2105.05233 [cs.LG].

[24] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. [Accessed: 19-04-2023]. 2015. arXiv: 1511.08458 [cs.NE].

[25] Kaiming He et al. *Deep Residual Learning for Image Recognition*. [Accessed: 19-04-2023]. 2015. arXiv: 1512.03385 [cs.CV].

[26] Valerio Biscione and Jeffrey S. Bowers. *Convolutional Neural Networks Are Not Invariant to Translation, but They Can Learn to Be*. [Accessed: 20-04-2023]. 2021. arXiv: 2110.05861 [cs.CV].

[27] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014). [Accessed: 20-04-2023], pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[28] Ekin D. Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. [Accessed: 31-03-2023]. 2019. arXiv: 1805.09501 [cs.CV].

[29] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. [Accessed: 31-03-2023]. 2019. arXiv: 1909.13719 [cs.CV].

[30] Sungbin Lim et al. *Fast AutoAugment*. [Accessed: 31-03-2023]. 2019. arXiv: 1905.00397 [cs.LG].

[31] Ryuichiro Hataya et al. *Faster AutoAugment: Learning Augmentation Strategies using Backpropagation*. [Accessed: 31-03-2023]. 2019. arXiv: 1911.06987 [cs.CV].

[32] Dan Hendrycks et al. *AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty*. [Accessed: 31-03-2023]. 2020. arXiv: 1912.02781 [stat.ML].

[33] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. [Accessed: 21-04-2023]. 2021. arXiv: 2103.00020 [cs.CV].

[34] Vaart Software (vaartsoftware.nl). *AFFiNe - Angling Freshwater Fish Netherlands*. [**non-commercial purposes only**] [Accessed: 24-04-2023]. URL: https://www.kaggle.com/datasets/jorritvenema/affine.

[35] Simo Ryu. *Low-rank Adaptation for Fast Text-to-Image Diffusion Fine-tuning*. [Accessed: 21-04-2023]. URL: https://github.com/cloneofsimo/lora.

[36] Daniel Bashir et al. *An Information-Theoretic Perspective on Overfitting and Underfitting*. [Accessed: 22-04-2023]. 2020. arXiv: 2010.06076 [cs.LG].

[37] Jason Brownlee. *How to Avoid Data Leakage When Performing Data Preparation*. [Accessed: 22-04-2023]. URL: https://web.archive.org/web/20230315012105/https://machinelearningmastery.com/data-preparation-without-data-leakage/.

[38] Charu C. Aggarwal. *Data Mining: The Textbook*. [Accessed: 22-04-2023]. URL: https://link.springer.com/book/10.1007/978-3-319-14142-8.

[39] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006). [Accessed: 22-04-2023], pp. 861–874. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010. URL: https://www.sciencedirect.com/science/article/pii/S016786550500303X.

[40] Bundesanstalt für Gewässerkunde / Federal Institute of Hydrology. *Supplementary dataset | 'BfG-Project Smart fish counter for monitoring species, size, migration behavior and environmental conditions' project*. [**any usage, copying, sharing, or publication of the data without consultation and approval from the BfG is prohibited**].

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[2]

I Aleksandr Ivanov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Overcoming Dataset Limitations: Advanced Augmentation Techniques for Fish Species Classification with Convolutional Neural Networks", supervised by Elizaveta Dubrovinskaya and Jeffrey Andrew Tuhtan
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

07.05.2023

[2]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Annotation tool

This additional material presents annotation tool User Interface (UI). This is a Jupyter Notebook widget that allows one to iterate over images and annotate them with bounding boxes. This tool was updated by the author, adding new UI features, better navigation, and additional background processing. The main core of is *jupyter-bbox-widget* available online[3]. On Figure 40 UI components of this tool are explained:

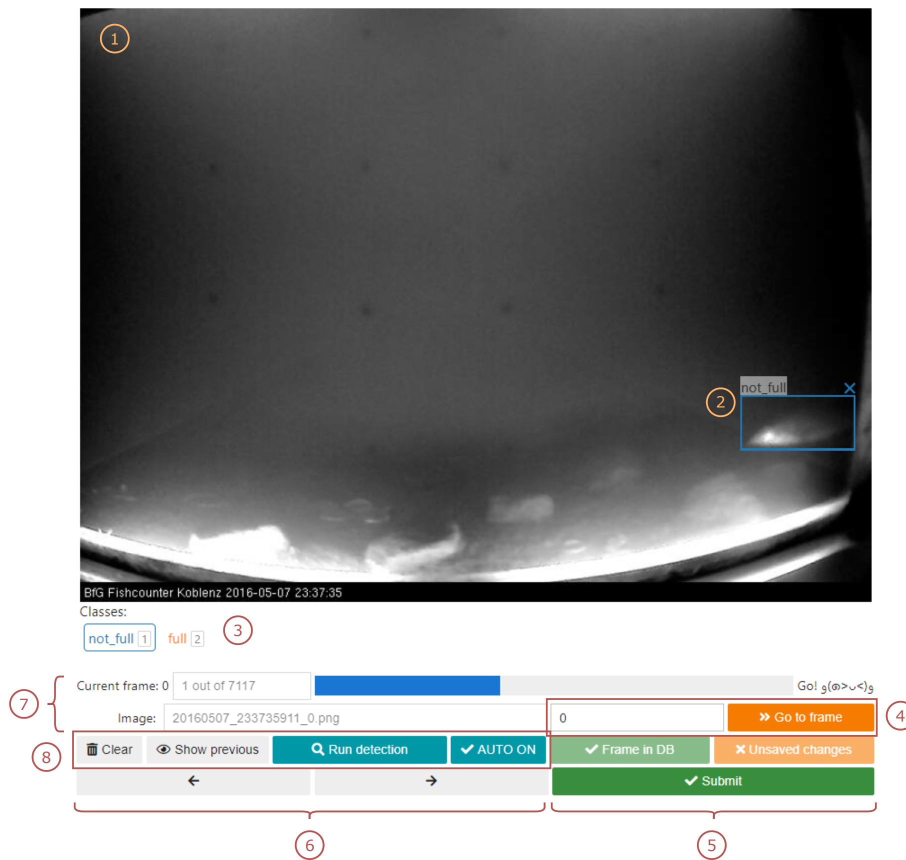


Figure 40. Annotation tool [40].

1. Frame preview
2. Bounding box around a fish with *not_full* class
3. Selection of active class
4. Fast navigation to a frame with a specific index
5. Frame database status and submit button
6. Previous and next frames navigation
7. Current frame index, name, and progress bar
8. Additional buttons: Remove bounding boxes, Show previous, Run build-in YOLO model to detect fish, Auto YOLO running on a new frame toggle

[3] https://github.com/gereleth/jupyter-bbox-widget

Appendix 3 – Caption tool

This additional material presents caption tool UI to manually create captions (description of an image) for further latent diffusion model training and finetuning. This tool is built on top of gradio[4] framework. After the user has selected the working directory, every image is loaded (on demand) and a corresponding *.txt* file is created. After that user can write a prompt for that image, or go to the different one. There is also a bulk image renaming tool, which automatically renames every image and corresponding caption file to a specified format. On Figure 41 UI components of this tool are explained:

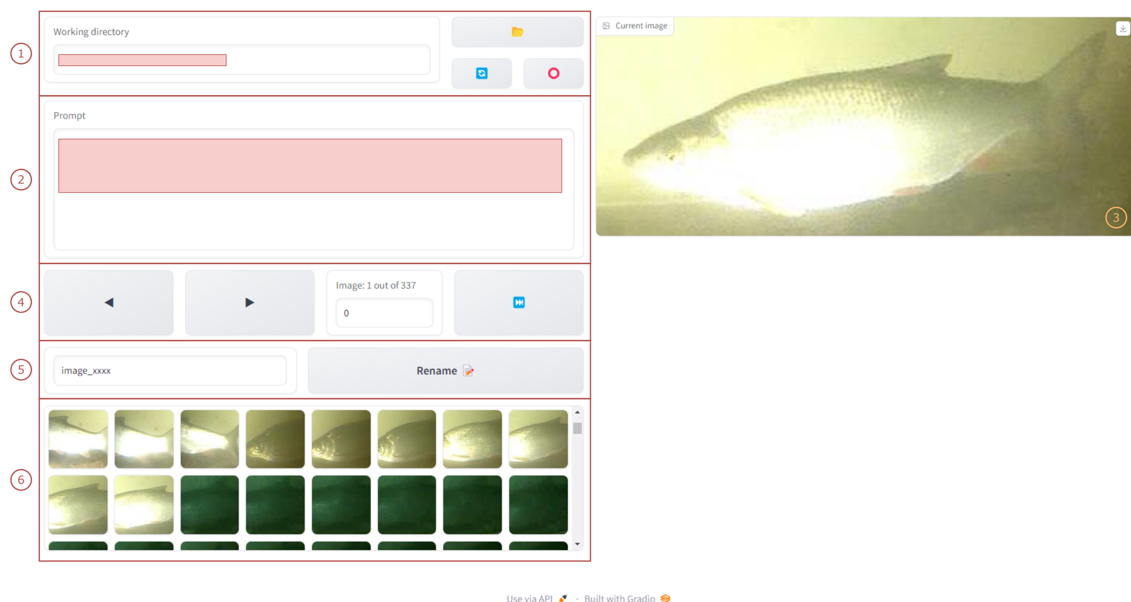


Figure 41. Image description caption tool [40].

1. Working directory selection. Additional buttons are *load* and *unload*
2. Prompt text area. Text is automatically saved to a caption file
3. Active image
4. Navigation toolbar
5. Bulk image renaming toolbar
6. Image gallery for fast and convenient selection

[4]https://gradio.app/docs/

Appendix 4 – Custom dataset class

Here is an implementation of a custom dataset class for model training in *Pytorch*.

The use of a custom dataset class gives the ability to control the form of image saving (comparing to *torchvision.datasets.ImageFolder*[5] where images must be physically stored and separated into *train* and *val* subfolders) and additional manipulation with data (in this case labels are converted into number-format with preservation of original labels + additional meta can be accessed, such as belonging to a specific subset)

```
1  class FishSpeciesDataset(Dataset):
2      def __init__(self, dataset_csv, images_root = "", transform=None, random_state=42):
3          self.data = pd.read_csv(dataset_csv)
4
5          self.img_paths = images_root + self.data['image_path'].to_numpy()
6
7          self.CLASSES, self.labels = np.unique(self.data['label'],
8              return_inverse=True)
9          self.labels = torch.tensor(self.labels)
10
11         self.transform = transform
12
13     def __len__(self, ):
14         return len(self.labels)
15
16     def __getitem__(self, index):
17         x = Image.open(self.img_paths[index])
18         y = self.labels[index]
19
20         if self.transform:
21             x = self.transform(x)
22
23         return x, y
```

[5] https://pytorch.org/vision/main/generated/torchvision.datasets.ImageFolder.html

Appendix 5 – Custom training loop

Here is an implementation of custom model training in *Pytorch*.

There are ready scripts available[6] for popular model training using *Pytorch* and *torchvision* libraries. However, utilizing a custom script gives a better understanding and control over what is happening under the hood.

```
1  def train_model(train_set, val_set, save2):
2
3      # ===================================
4
5      model = get_model()
6
7      loss_fn = nn.CrossEntropyLoss()
8      optimizer = torch.optim.Adam(model.parameters(), lr=LR)
9      scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=MAX_EPOCH)
10
11     # ===================================
12
13     pbar = tqdm(total=MAX_EPOCH)
14
15     train_loss = np.zeros(MAX_EPOCH)
16     train_acc = np.zeros(MAX_EPOCH)
17     val_acc = np.zeros(MAX_EPOCH)
18     lr = np.zeros(MAX_EPOCH)
19
20     train_loader = get_loader(train_set)
21     val_loader = get_loader(val_set)
22
23     train_batches = len(train_loader)
24     val_batches = len(val_loader)
25
26     for epoch in range(MAX_EPOCH):
27
28         train_lossx = np.zeros(train_batches)
29         train_accx = np.zeros(train_batches)
30         val_accx = np.zeros(val_batches)
31
32         # =============== train cycle ===============
33
34         model.train()
35
36         for batch_idx, (X, y) in enumerate(train_loader):
37             X_c = X.cuda(non_blocking=True)
38             y_c = y.cuda(non_blocking=True)
39
40             raw = model(X_c)
```

[6]https://github.com/pytorch/vision/tree/main/references/classification

```
41              loss = loss_fn(raw, y_c)
42
43              optimizer.zero_grad()
44              loss.backward()
45              optimizer.step()
46
47              pred = raw.argmax(1)
48
49              train_accx[batch_idx] = (torch.sum(pred==y_c) / len(pred)).cpu()
50              train_lossx[batch_idx] = loss.item()
51
52          scheduler.step()
53
54          # ================ save model ================
55
56          if (epoch % SAVE_EVERY) == 0:
57              torch.save(model, os.path.join(save2, f'model_{epoch}.pt'))
58
59          # ================ val cycle ================
60
61          model.eval()
62
63          with torch.inference_mode():
64              for batch_idx, (X, y) in enumerate(val_loader):
65                  X_c = X.cuda(non_blocking=True)
66                  y_c = y.cuda(non_blocking=True)
67
68                  pred = model(X_c).argmax(1)
69
70                  val_accx[batch_idx] = (torch.sum(pred==y_c) / len(pred)).cpu()
71
72          # ================ info ================
73
74          mean_loss = np.mean(train_lossx)
75          mean_tacc = np.mean(train_accx)
76          mean_vacc = np.mean(val_accx)
77
78          # update statistics
79          train_loss[epoch] = mean_loss
80          train_acc[epoch] = mean_tacc
81          val_acc[epoch] = mean_vacc
82          lr[epoch] = optimizer.param_groups[0]["lr"]
83
84          # update bar
85          pbar.update()
86          pbar.set_description(f'Epoch {epoch+1}')
87          pbar.set_postfix({'loss': f'{mean_loss:.4E}',
88                            'train acc': f'{mean_tacc:.2f}',
89                            'val acc': f'{mean_vacc:.2f}',
90                            'lr': f'{optimizer.param_groups[0]["lr"]:.2E}'})
91
92      return (pd.DataFrame({
93          "epoch": np.arange(MAX_EPOCH) + 1, "lr": lr,
94          "loss": train_loss, "train acc": train_acc,
95          "val acc": val_acc}), model, optimizer, scheduler)
```

Appendix 6 – SD Supplementary fine-tuning dataset

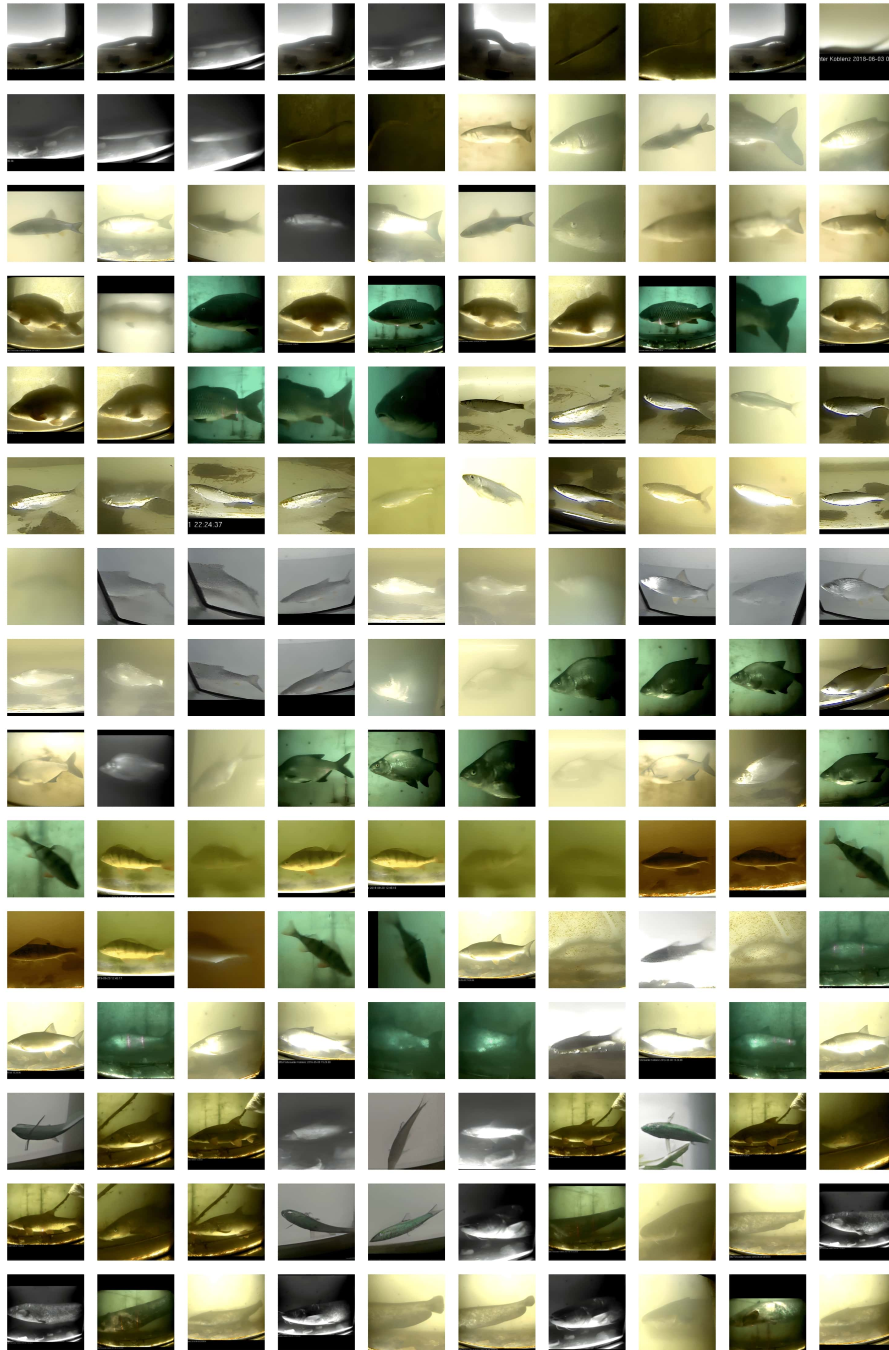


Figure 42. Supplementary images used for fine-tuning Stable Diffusion [40].

Appendix 7 – SD AFFiNe fine-tuning dataset

Figure 43. AFFiNe images used for fine-tuning Stable Diffusion [34].

Appendix 8 – RunPod post-installation script

This script was manually inserted into the remote console with one reconnecting to initialize *conda* system. Answers to "*accelerate config*" were *[This machine, No distributed training, NO, NO, NO, 0, bf16]*.

```
cd /workspace
apt-get update

git clone https://github.com/bmaltais/kohya_ss.git
cd /workspace/kohya_ss

apt install python3-tk -y
pip install -U pip setuptools
apt-get install ffmpeg libsm6 libxext6 unzip -y

mkdir -p /workspace/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh \
-O /workspace/miniconda3/miniconda.sh
bash /workspace/miniconda3/miniconda.sh -b -u -p /workspace/miniconda3
rm -rf /workspace/miniconda3/miniconda.sh
/workspace/miniconda3/bin/conda init bash
/workspace/miniconda3/bin/conda init zsh

cd /workspace/kohya_ss

conda create --name kohya_ss python=3.10 -y
conda activate kohya_ss

apt install python3-tk -y
pip install -U pip setuptool

pip install torch==1.12.1+cu116 torchvision==0.13.1+cu116 \
--extra-index-url https://download.pytorch.org/whl/cu116
pip install --use-pep517 --upgrade -r requirements.txt
pip install -U -I --no-deps https://github.com/C43H66N12O12S2/stable-diffusion-\
webui/releases/download/linux/xformers-0.0.14.dev0-cp310-cp310-linux_x86_64.whl
conda install cudatoolkit -y

accelerate config
```

Appendix 9 – Checkpoint selection

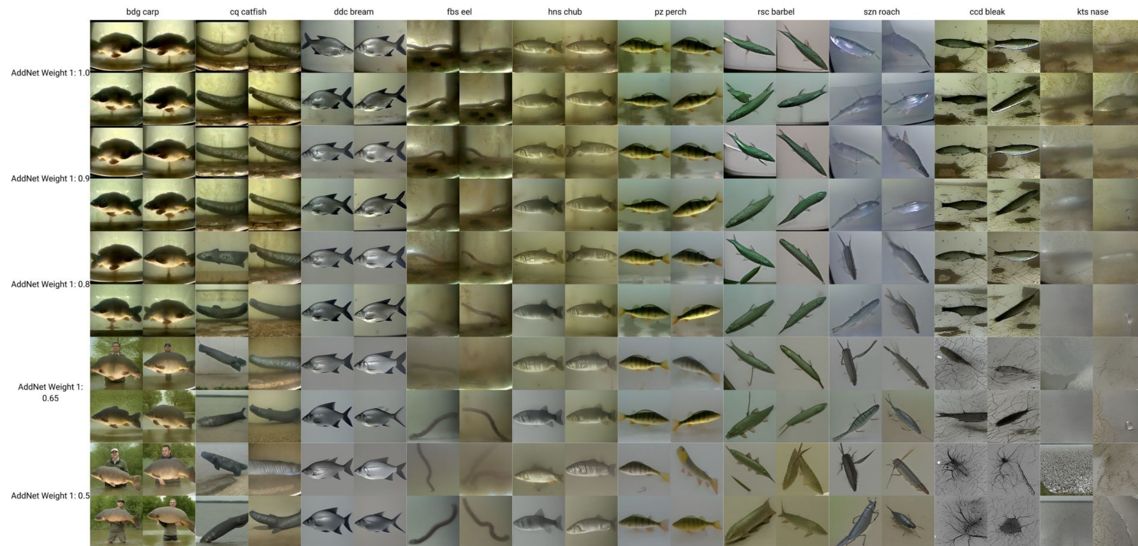


Figure 44. SD v1.5 *Supplementary* checkpoint analysis (*LoRA*).

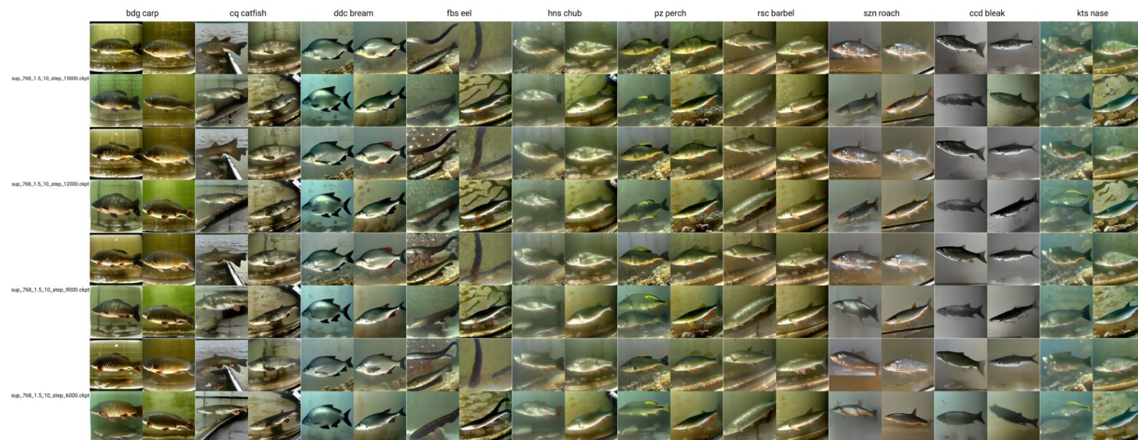


Figure 45. SD v1.5 *Supplementary* checkpoint analysis (*DreamBooth*).

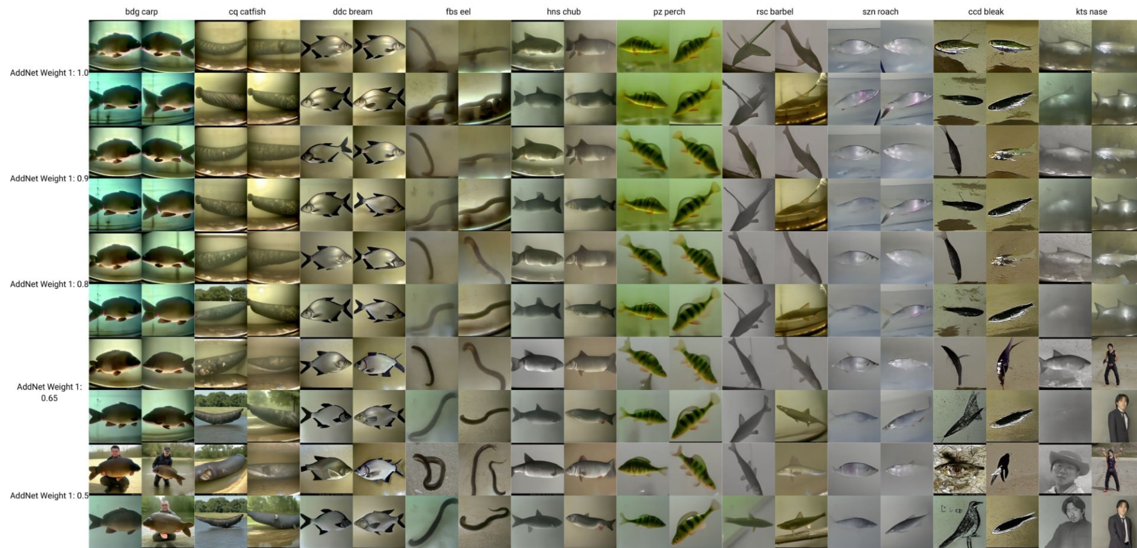


Figure 46. SD v2.1 *Supplementary* checkpoint analysis (*LoRA*).

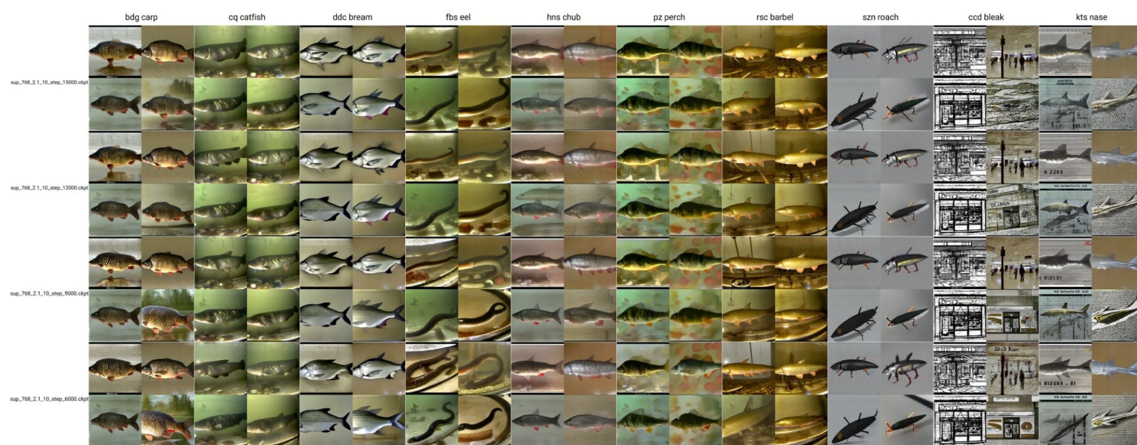


Figure 47. SD v2.1 *Supplementary* checkpoint analysis (*DreamBooth*).

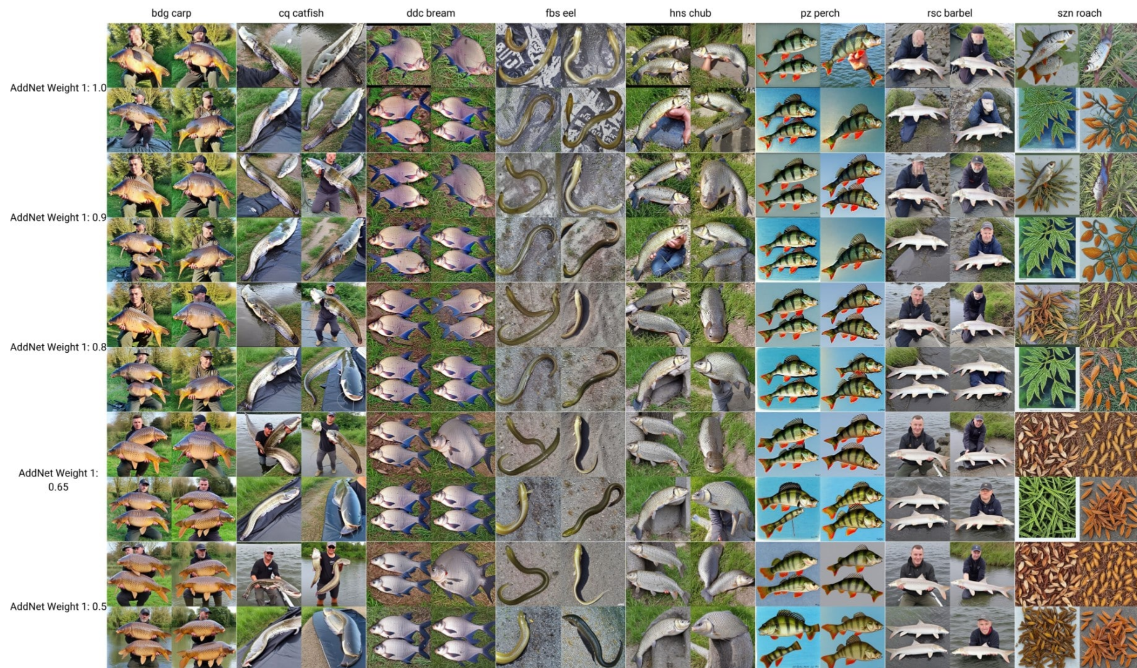


Figure 48. SD v1.5 *AFFiNe* checkpoint analysis (*LoRA*).

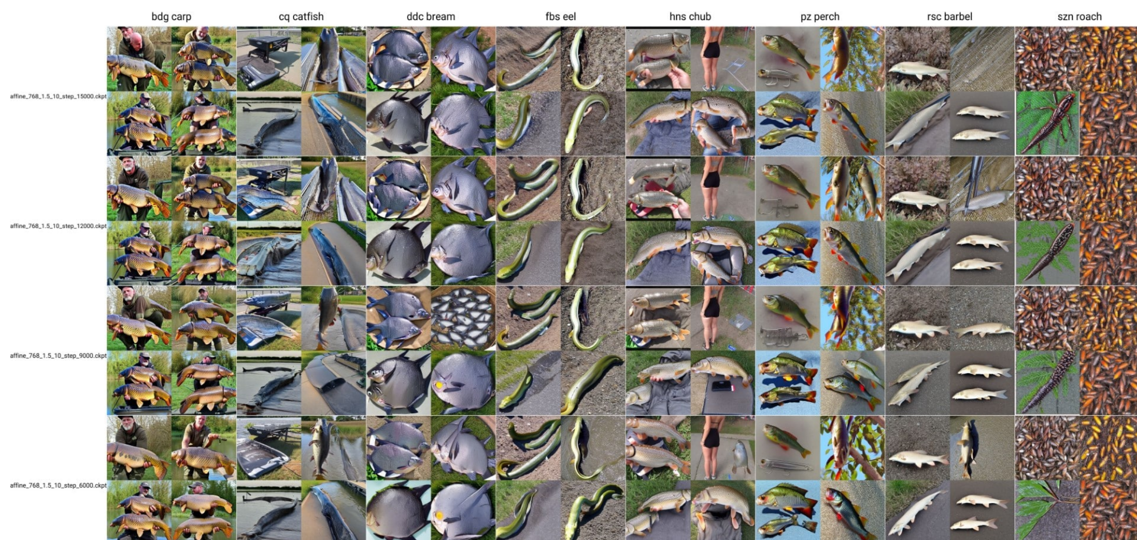


Figure 49. SD v1.5 *AFFiNe* checkpoint analysis (*DreamBooth*).

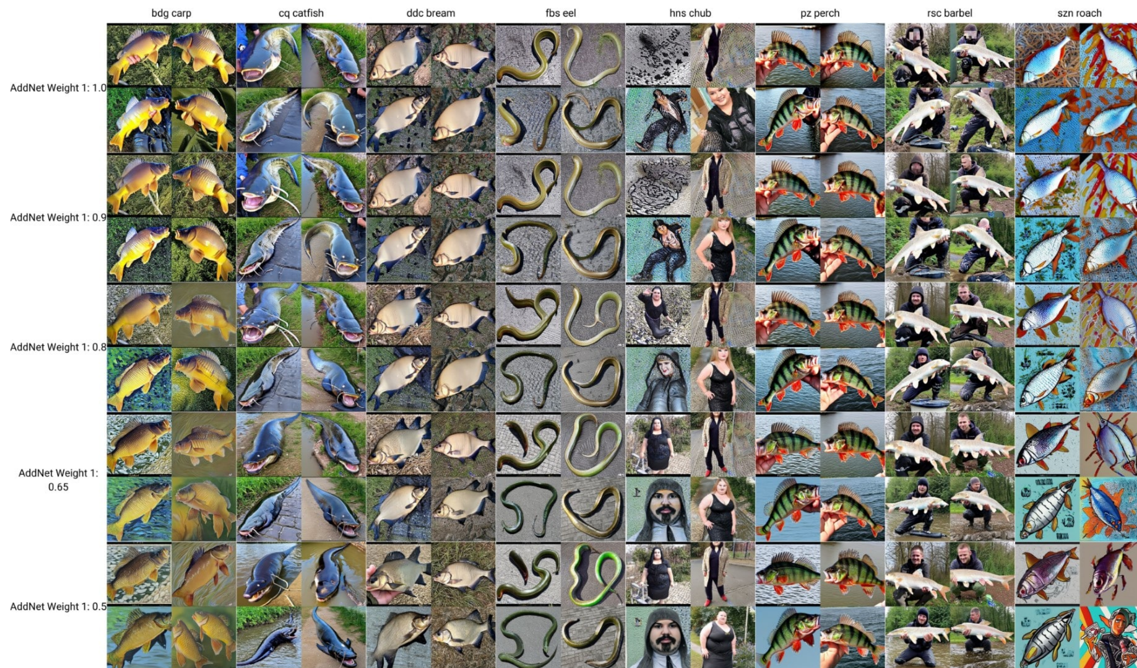


Figure 50. SD v2.1 *AFFiNe* checkpoint analysis (*LoRA*).

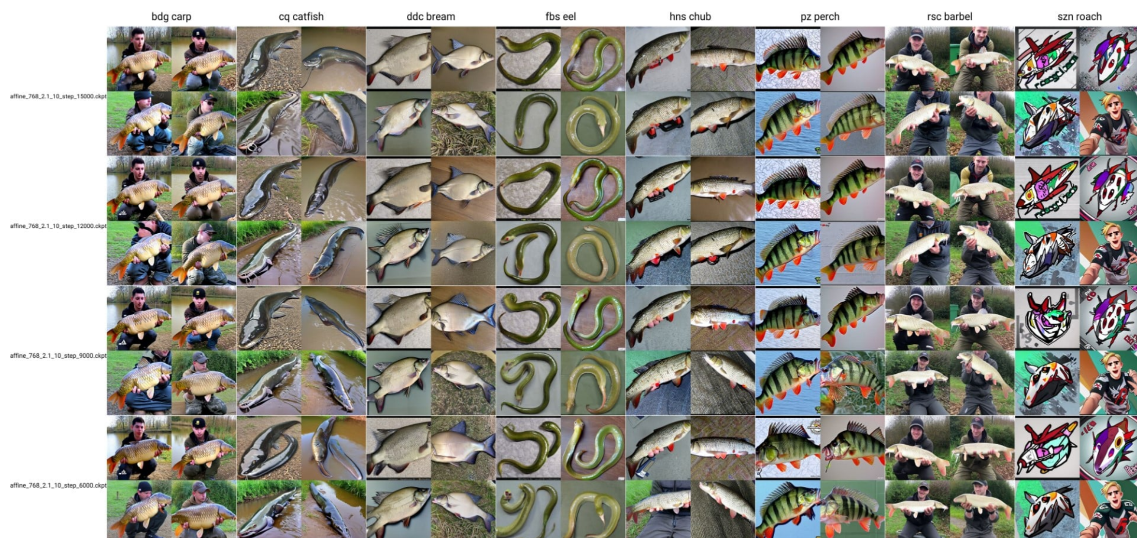


Figure 51. SD v2.1 *AFFiNe* checkpoint analysis (*DreamBooth*).

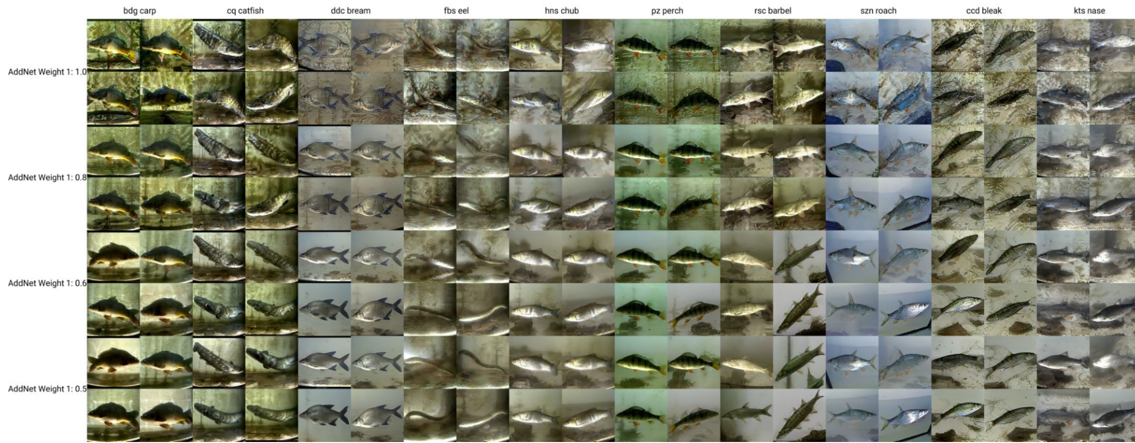


Figure 52. SD v1.5 *Supplementary+AFFiNe* checkpoint analysis (*LoRA*).

Figure 53. SD v1.5 *Supplementary+AFFiNe* checkpoint analysis (*DreamBooth*).

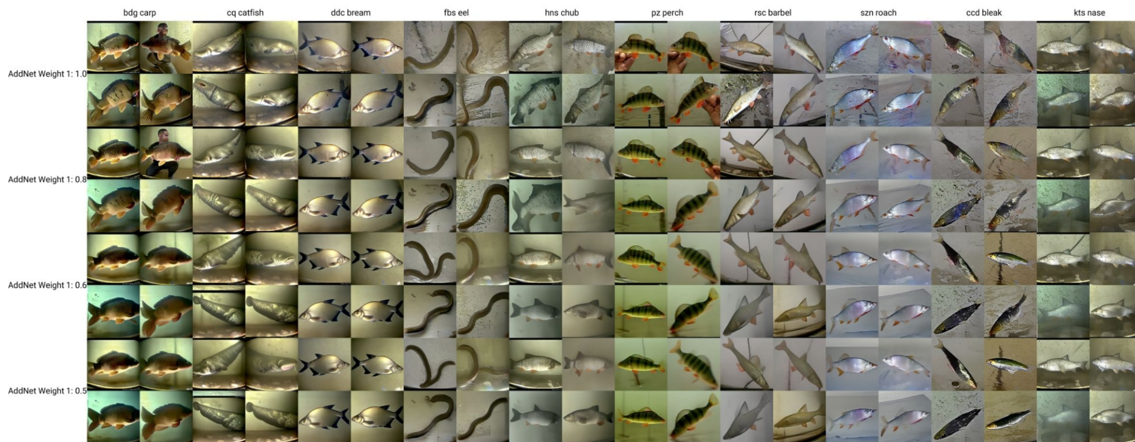


Figure 54. SD v2.1 *Supplementary+AFFiNe* checkpoint analysis (*LoRA*).

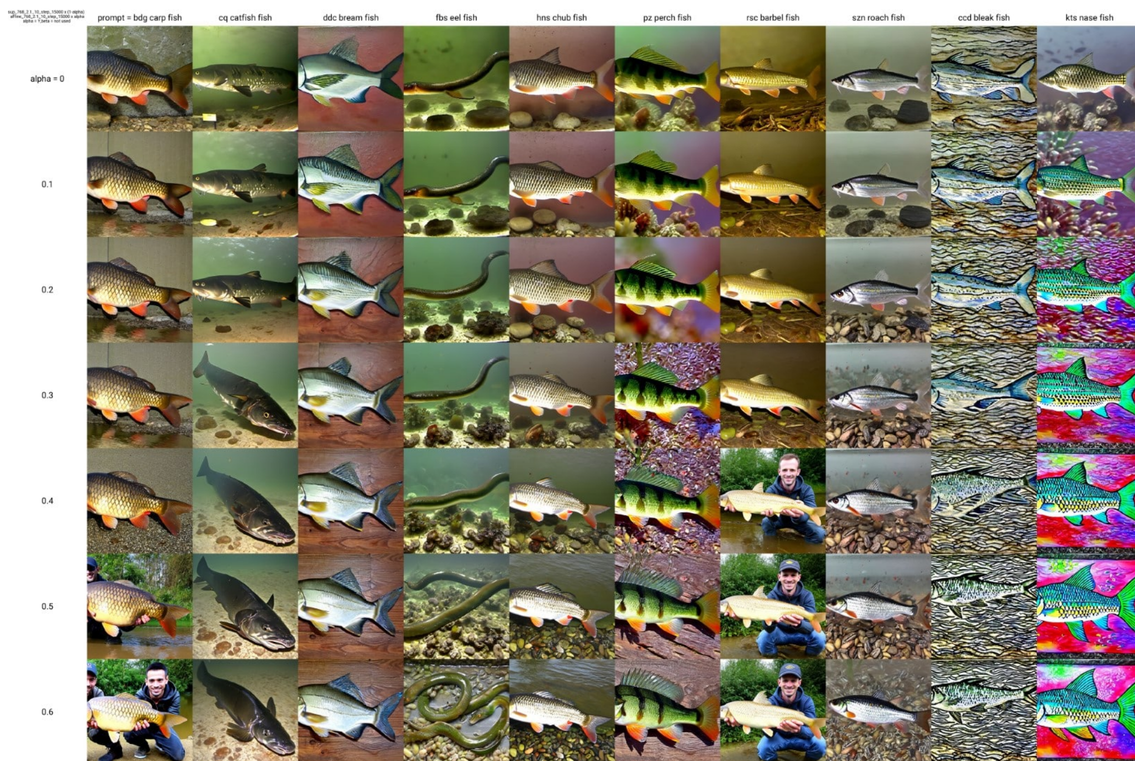


Figure 55. SD v2.1 *Supplementary+AFFiNe* checkpoint analysis (*DreamBooth*).

Appendix 10 – Synthetic datasets

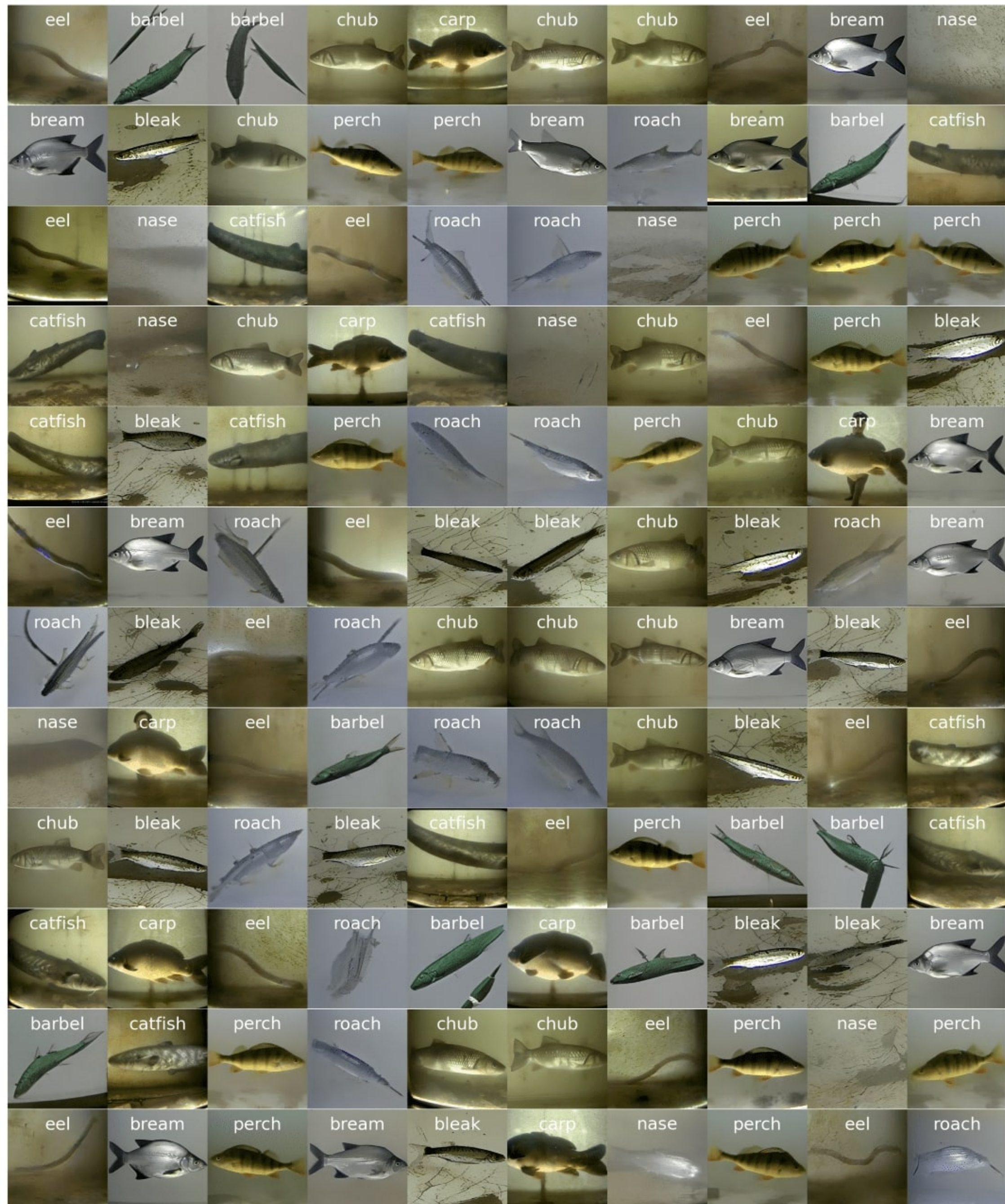


Figure 56. SD v1.5 *Supplementary* synthetic images (*LoRA*).

Figure 57. SD v1.5 *Supplementary* synthetic images (*DreamBooth*).

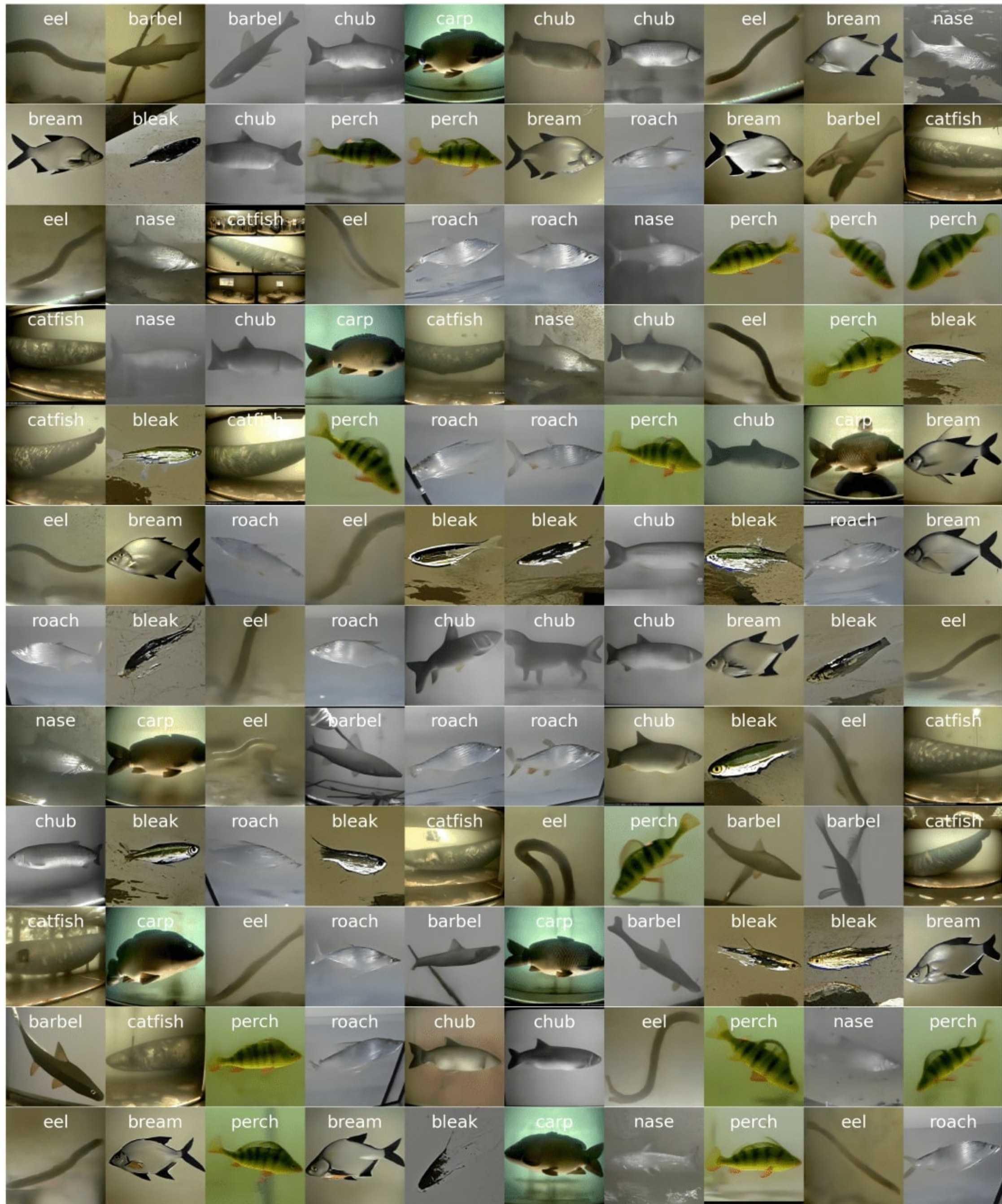


Figure 58. SD v2.1 *Supplementary* synthetic images (*LoRA*).

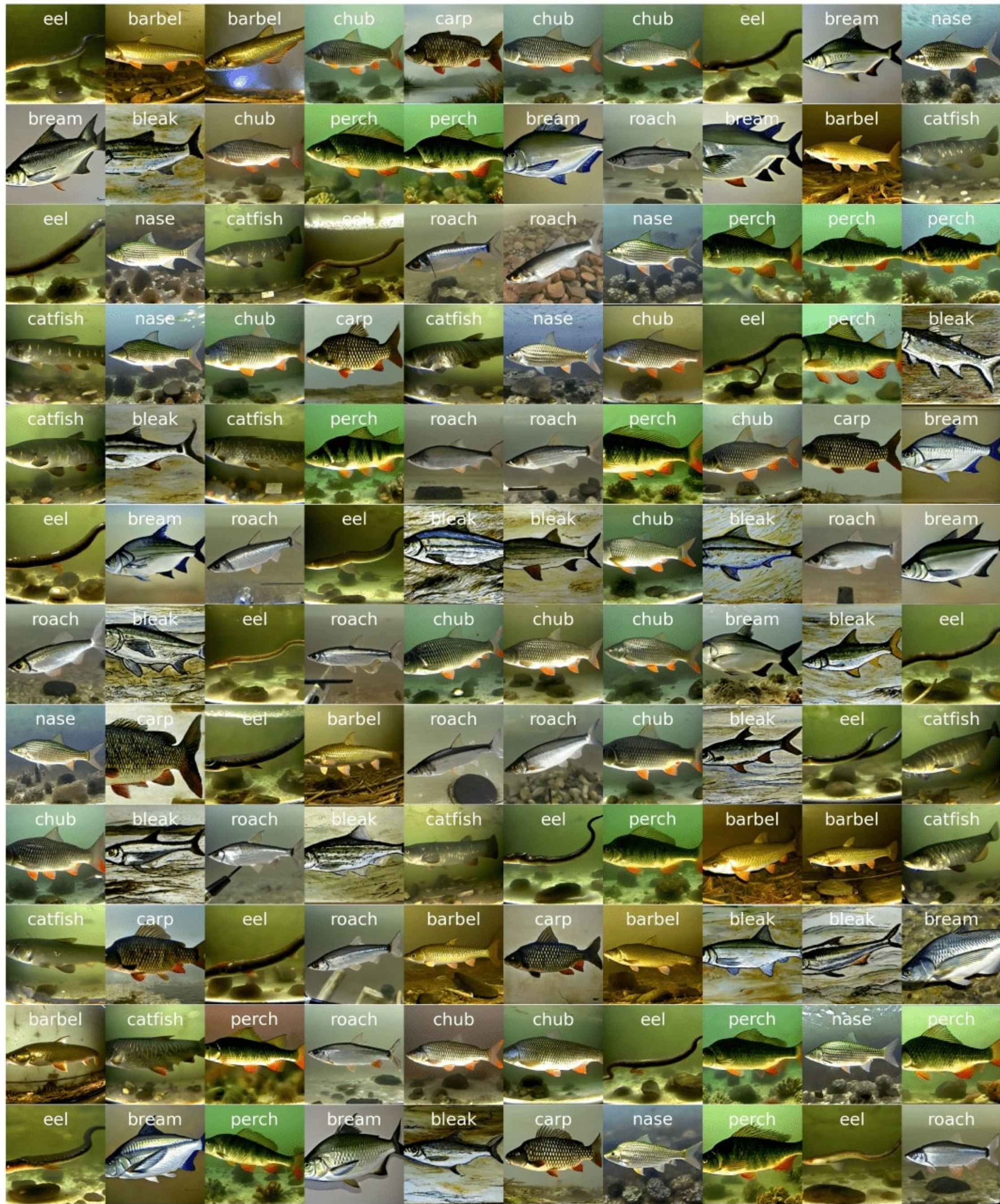


Figure 59. SD v2.1 *Supplementary* synthetic images (*DreamBooth*).

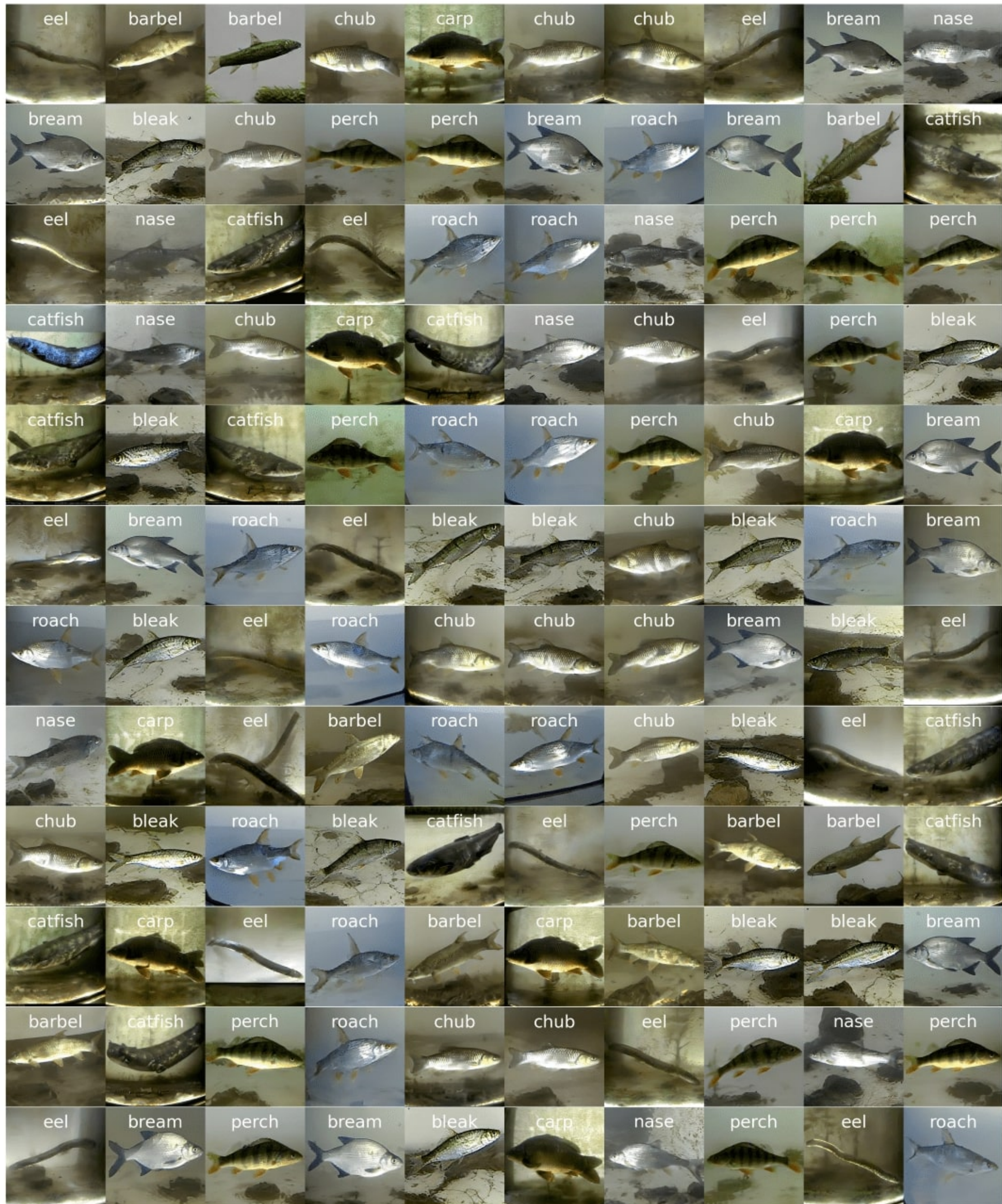


Figure 60. SD v1.5 *Supplementary+AFFiNe* synthetic images (*LoRA*).

Figure 61. SD v1.5 *Supplementary+AFFiNe* synthetic images (*DreamBooth*).

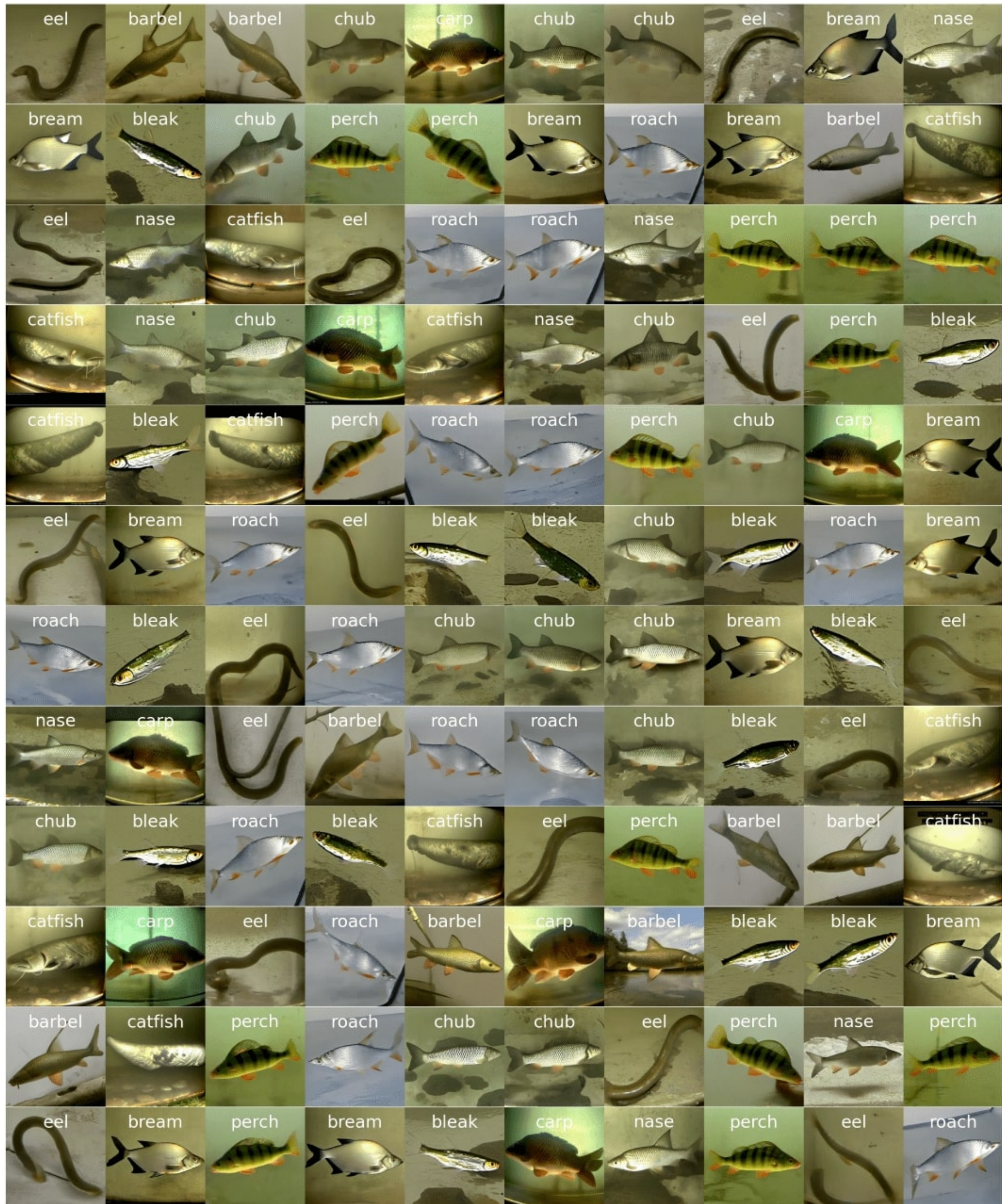


Figure 62. SD v2.1 *Supplementary+AFFiNe* synthetic images (*LoRA*).

Figure 63. SD v2.1 *Supplementary+AFFiNe* synthetic images (*DreamBooth*).