Progan - Progressive Growing of GANs

Valérie Alicia Witt, Zoé Loschen, Philip Schwientek, Manuel Kuhn

29.06.22

1 ProGAN - an Overview

ProGAN is a generative adversarial network (GAN) developed by NVIDIA [1]. Fundamentally, GANs are generative models designed to produce new data that closely resembles the training data, such that the generated samples are indistinguishable from the originals. To achieve this, a generator is trained to synthesize new samples from random inputs, while a discriminator is trained to assign a probability to each sample, reflecting how likely it is to belong to the training set. During training, both the generator and discriminator are optimized simultaneously. The generator's goal is to produce data convincing enough to deceive the discriminator, whereas the discriminator aims to correctly distinguish generated samples from real ones.
In GANs, the loss does not directly indicate the overall performance of the network but instead reflects the relative success of the generator and discriminator in their opposing tasks.
Training GANs, however, can be quite challenging. In general, the process is prone to instability [2].
ProGAN introduces a progressive training approach for image generation. Both the generator and discriminator begin at a low resolution, which is gradually increased by adding layers. As training progresses, the generated images become more detailed. This staged approach allows the model to learn coarse structures first, with finer details added incrementally at higher resolutions. A substantial portion of training occurs at lower resolutions, where variance is limited—this helps to stabilize the training. Moreover, training iterations run significantly faster at lower resolutions, contributing to an overall more efficient process.

2 Task Description

Our project is based on an existing project from *GitHub*[3] and it is supposed to generate new laundry images based on a dataset of laundry images on a laundry belt.

3 Implementation/Methods

3.1 Refactoring

Since the GitHub project is based on a Jupyter Notebook file, the existing code had to be split into logical sections and moved into separate Python files in order to achieve a cleaner and more maintainable structure. The refactored code now includes a *config.py*, a *discriminator.py*, a *generator.py*, and a *main.py* file, allowing a more structured and efficient workflow.

3.2 Padding

As the training images no longer matched the resolution used in the project when generating higher resolution output, a few rows of black pixels were added to the upper and lower edges of

the training images.

3.3 Hyperparameters

Finding suitable hyperparameters is a common challenge when working with GANs, as unsupervised training lacks clear, automated evaluation metrics. Tools like *KerasTuner* [4], which rely on performance metrics to tune models, are primarily designed for supervised learning scenarios.

3.4 Color Variation

3.4.1 Adjusting the Dataset

A large portion of the dataset provided consists of images showing white laundry items. This is due to the fact that nursing homes and hospitals, which are major clients of laundries, tend to generate predominantly white laundry.
In order to enable the generation of images of laundry items in various colors, the neural network must be trained on a correspondingly diverse dataset. Therefore, a part of the dataset with available segmentation masks was artificially colored in red, green, white, and blue.
The recoloring process modifies the hue angle h of the pixels within the segmentation mask to match the target color: green: $h = 1/3$, red: $h = 0$, blue: $h = 2/3$ (with $h \in [0, 1]$). White coloring is achieved by reducing the saturation by half.

3.4.2 Interpolating the Color Space

Interpolating within the color space, can be achieved by determining the encoding of the laundry's color in the *latent space*. Our approach is based on calculating a prototype vector for each color, representing that color's encoding in the latent space. This prototype vector is computed as the average of several latent vectors ($n = 100$) corresponding to images of the same color.
To change the color of a generated laundry image from an original color 1 with prototype vector p_1 to a new color 2 with prototype vector p_2, we use the following formula:

$$l_{new} = l_{old} - p_1 + p_2$$

l_{new} serves as the input to the ProGAN generator, which then produces the same laundry item in the new color.
As aforementioned, multiple images per color ($n = 100$) need to be generated to compute the prototype vectors. Since manually classifying 400 or more images by color would be highly time-consuming, we implemented an automated classification based on pixel color distributions. If a color exceeds a certain pixel threshold and dominates the image, that image and its latent vector are used in calculating the corresponding prototype.

3.4.3 Interpolation Between Latent Space Vectors

In order to explore the internal structure of the latent space, interpolation is conducted between two latent space vectors and images are generated from the interpolated values. This latent structure is assumed to be preserved in the resulting images. The interpolation itself is computed as follows:
Let v_0 be the starting vector, v_1 the target vector, and v_t the interpolated vector, with $t \in [0, 1]$ Then:

$$v_t = v_0 + t \cdot (v_1 - v_0)$$

To visualize this interpolation, a script was developed that generates an animated GIF from two latent vectors (identified via the pseudo-random seed — vectors can be randomly generated through the UI, and the seed is displayed), the number of frames n, and the delay d between frames.
The script computes n evenly spaced values for t as:

$$t = \frac{k}{n-1} \quad \text{with} \quad k \in \{0, 1, ..., n-1\}, n \geq 2$$

For each value of t interpolated latent vectors are calculated and used for image generation. These frames are then compiled into a GIF animation.

3.4.4 Conditional GAN (CGAN)[5]

Another approach we explored to generated colored laundry images was the use of *Conditional GANs* (CGANs), a subclass of GANs.
Our implementation was based on a Kaggle project.
Since CGANs require images labeled with their corresponding color, we changed our dataset accordingly.
Due to memory errors emerging when training on full-resolution images, we downscaled all input images to a resolution of 270×180 pixels.
Figures 6–9 in the image appendix show the training results for various resolutions. While the generation of the correct color was successful, the shape of the laundry item remained indistinct, even at higher resolutions. Because the other approach had already produced better results by this point, we discontinued the CGAN experiments.

3.5 Creating a Graphical Interface

To enable users to interact with the results of this project and generate a wide range of images, a graphical user interface was developed.
A web-based interface was chosen due to its ease of implementation. Bootstrap 5 facilitated the straightforward design of input and output components, while JavaScript efficiently connected

user interactions with backend processes. Provided that image generation requests are not made in excessive quantities simultaneously, this setup delivers fast and reliable results.

3.5.1 Graphical Interface for Image Generation

To allow quick and simple image creation, a graphical interface was implemented that generates an image at the selected resolution with a single button click. This functionality is provided through Flask [6], a lightweight web framework that serves the generated images via dynamically constructed URLs.
Each URL includes a string identifying the function to be executed, followed by the desired resolution and a seed value to ensure image variability. Users can freely set both the seed and resolution, enabling the generation of diverse outputs.
The interface itself is built using basic HTML, while JavaScript handles dynamic frontend behavior such as updating the displayed image. When the "Generate Image" button is clicked, the URL is assigned to the *src* attribute of the corresponding HTML `img` element.
The interface was further enhanced to support color manipulation through operations in the latent space. Users can now choose a color for the generated image, which is passed as an additional parameter in the URL and used by the backend function.

3.5.2 Graphical Interface for Image Interpolation

Beyond image generation, a separate interface was developed to visualize the interpolation between two images. This view displays three images side by side: Two reference images generated from different seeds on the left and right, and an interpolated image in the center. Users select a common resolution and input individual seed values for the two outer images, which are generated independently.
A slider controls the interpolation process. When positioned fully to the left, the center image matches the left one; fully to the right, it mirrors the right image. Sliding it gradually causes a smooth transition, morphing the center image from the left to the right image. Depending on the input, both the shape and color of the image can change during interpolation.

3.6 Experiments

The ProGAN model was trained on both the original dataset and the recolored dataset using an NVIDIA GPU. The model was implemented using the TensorFlow framework [7]. As outlined in Section 2, training was conducted iteratively with increasing resolutions: 5×3, 10×6, 20×12, 40×24, and 80×48. For higher resolutions (160×96, 320×192, 640×384, and 1280×768), a more powerful GPU than was available for this project would be required. The model was trained for 40 epochs per resolution on the original dataset, and 20 epochs per resolution on the recolored dataset.

4 Results

Figure 1 shows the training results on the original dataset at the highest achieved resolution of 48×80. The background, including the conveyor belt and side walls, is clearly visible, and the laundry items themselves are well distinguishable. However, the resolution is still too low to show fine details, such as a checkered pattern. It is noticeable that all laundry items generated by this model appear white or gray.
Figure 2 gives two examples of recolored laundry items, and Figure 5 displays results generated by the model trained on the recolored dataset. It is evident that the network can generate laundry in various distinct colors, matching those seen in the training data (cf. Figure 2). However, it is also apparent that the model sometimes produces multi-colored laundry items.
An example of latent space interpolation as described in Section 3.4.2 is shown in Figure 4. In general, the interpolation method works reasonably well, though there are cases where it fails to produce coherent results.

5 Discussion and Conclusion

As discussed in Section 3.4, the original training data predominantly consists of white laundry items, with few colored examples. This likely led the trained generator to produce only white laundry items. In contrast, the model trained on the recolored dataset was able to generate laundry in red, blue, white, and green (cf. Section 3.6). This suggests that a balanced dataset is necessary to train a model capable of generating a diverse range of colors. While the recoloring method introduced here can produce a color-diverse dataset from existing data, generating realistically colored laundry likely requires using the original colors. Additionally, recoloring requires segmentation masks, which must be manually created. It would therefore be beneficial to collect more training images with a wide variety of colors.
The unsatisfying results regarding color variation likely stem from the generation of multi-colored items. The assumption behind the interpolation approach (cf. Section 3.4.2) is that there exists a location in latent space where the color is encoded, represented by the prototype vector. If, however, a single latent vector can result in a multi-colored image, this contradicts that assumption and undermines the method. An alternative could be the use of *conditional GANs* (cf. Section 3.4.4), which explicitly encode attributes like color. However, this method requires a properly labeled dataset.
A broader challenge beyond the scope of this project is the generation and interpolation of multi-colored laundry items — a necessary step toward generating a wide variety of real-world laundry appearances.
Finally, for a robust evaluation of the dataset, model, and chosen hyperparameters, it is essential to train the model to the target resolution. Unfortunately, the available hardware was insufficient to reach this goal within the project scope.

6 Image appendix

Abbildung 1: Generated images with resolution 48x80.

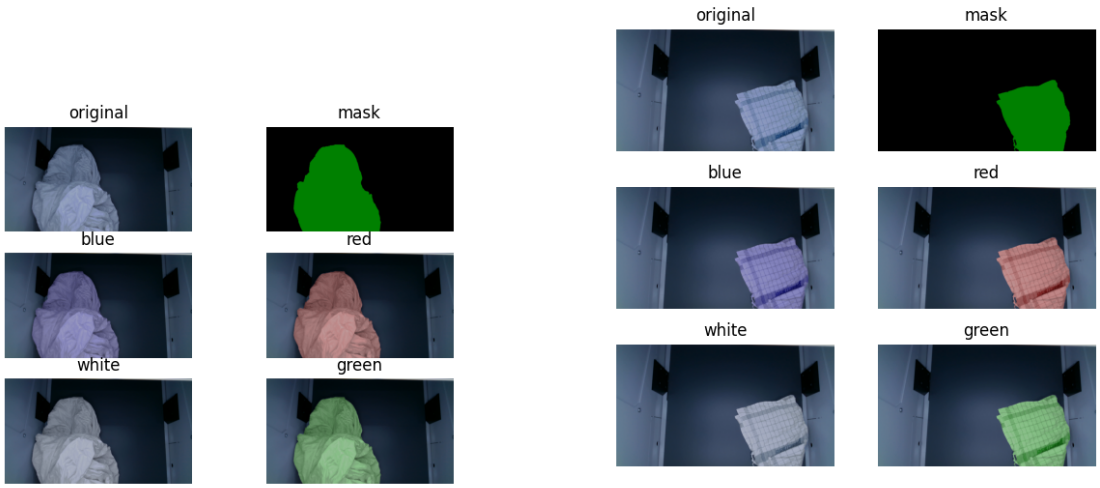

Abbildung 2: Colored laundry items and their mask.

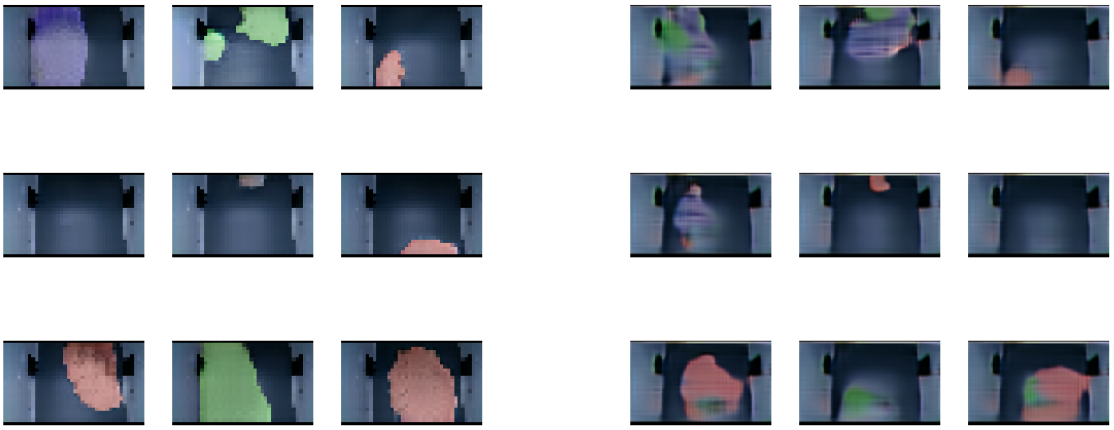

Abbildung 3: Generated images with resolution 24x40 and 48x80.

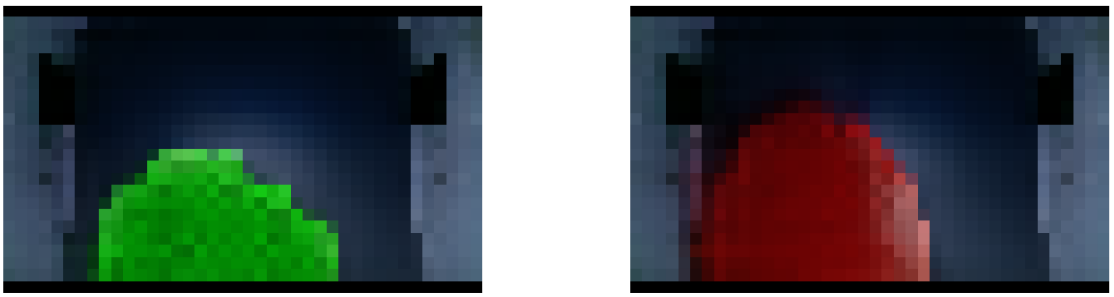

Abbildung 4: Original image (left), image colored in red via interpolation (right).

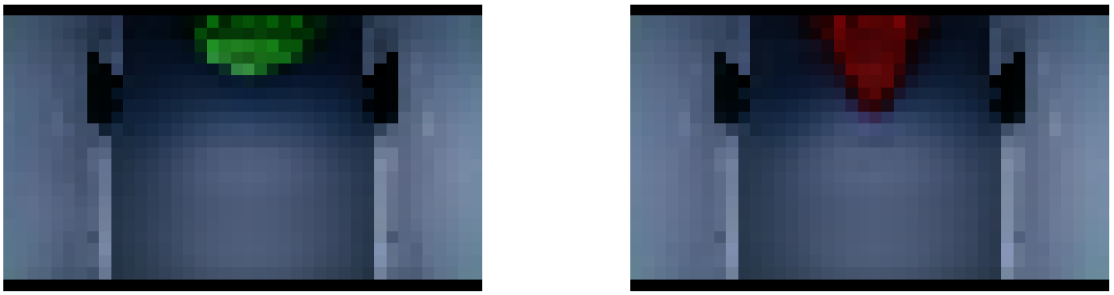

Abbildung 5: Generated images for the prototype vectors corresponding to the colors green and red.

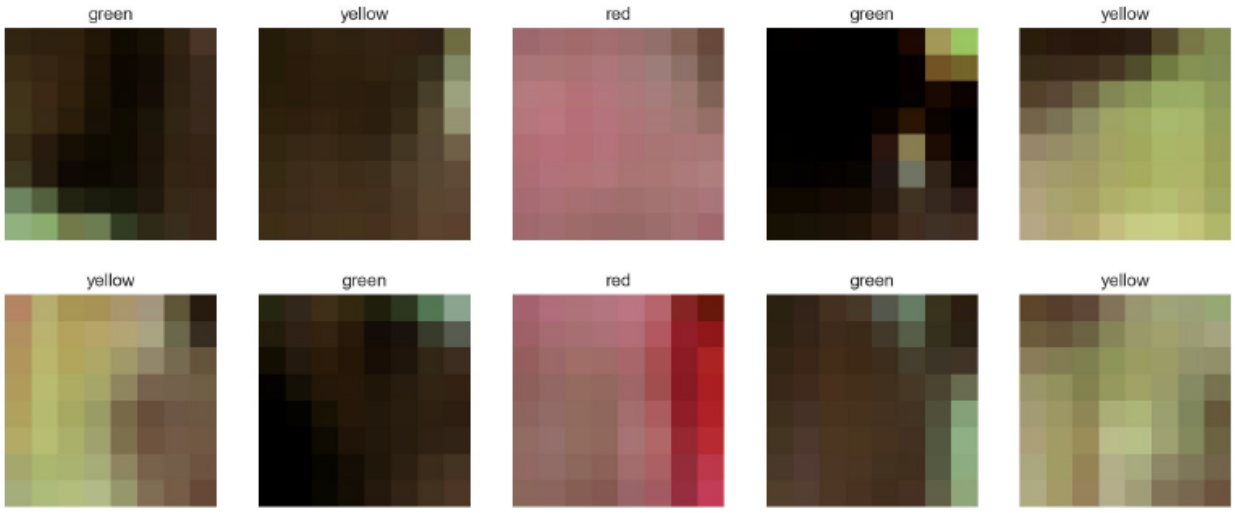

Abbildung 6: Output provided by CGAN

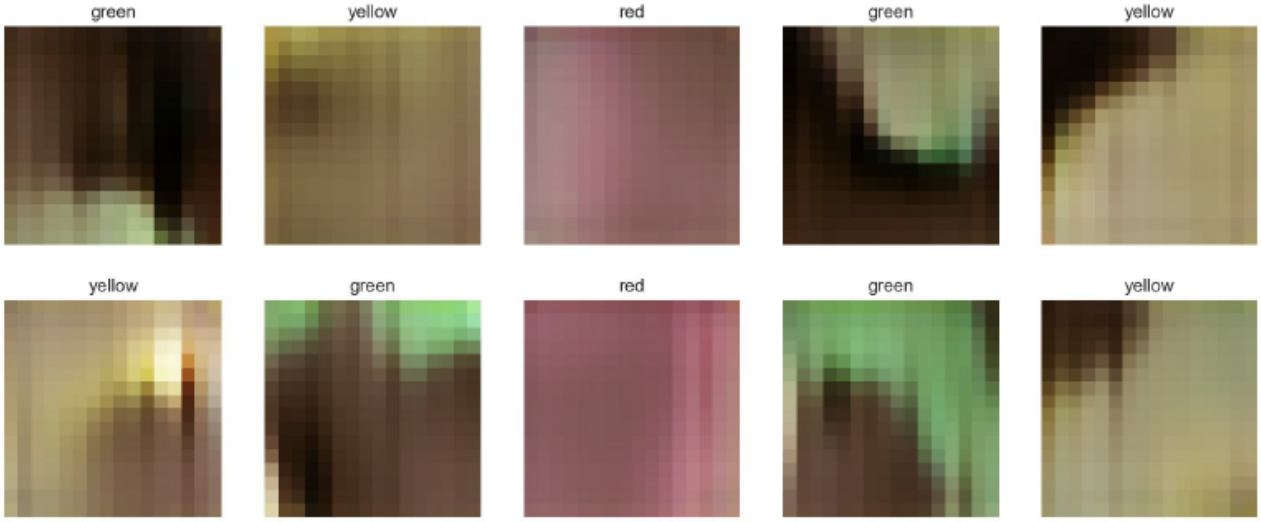

Abbildung 7: Output provided by CGAN

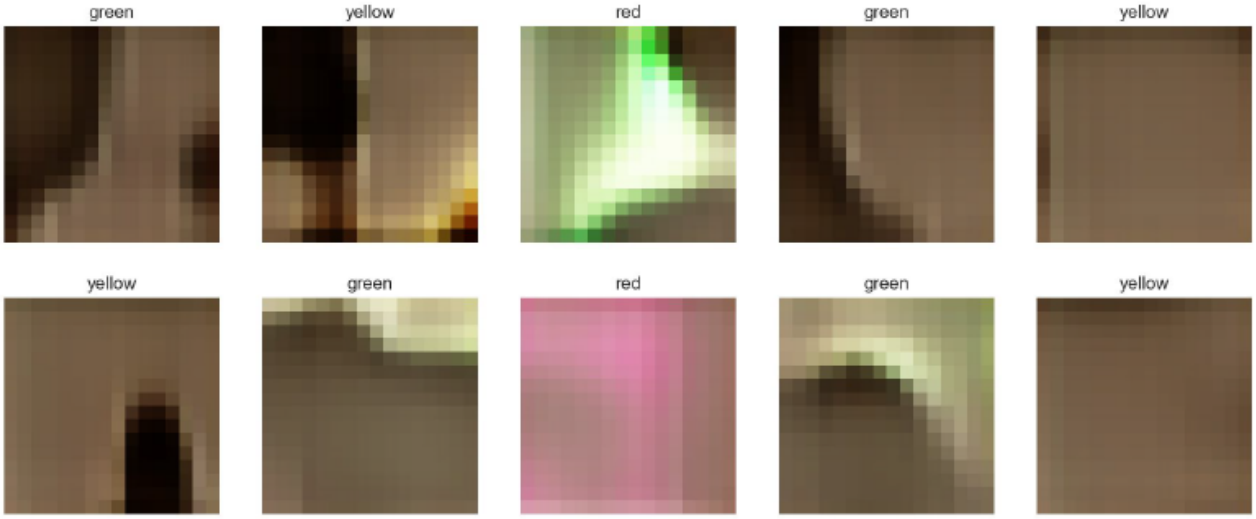

Abbildung 8: Output provided by CGAN

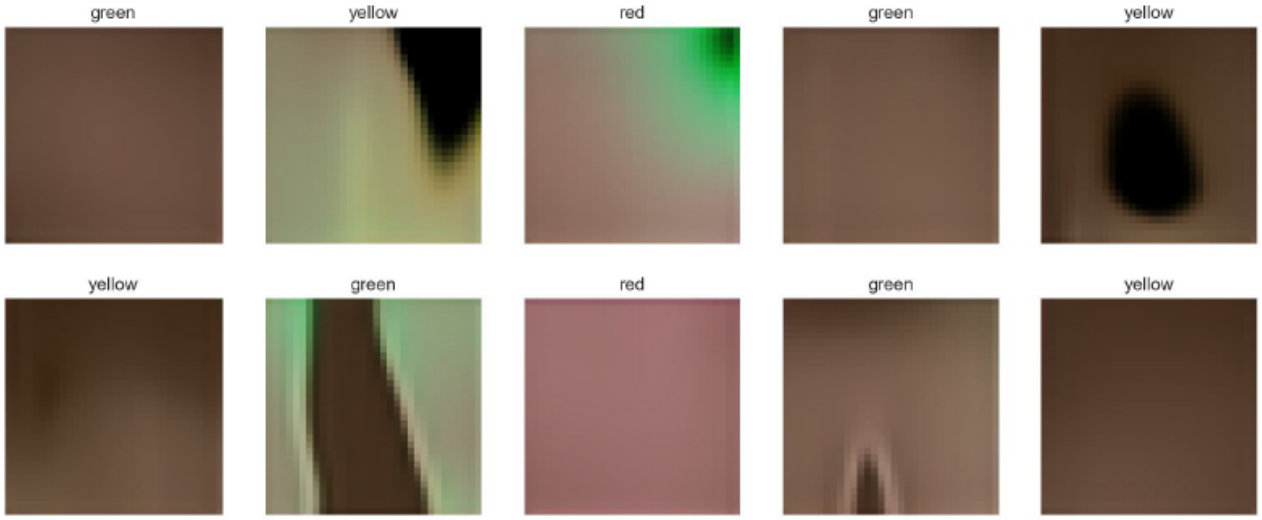

Abbildung 9: Output provided by CGAN

Literatur

[1] Tero Karras u. a. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *CoRR* abs/1710.10196 (2017). arXiv: 1710.10196. URL: http://arxiv.org/abs/1710.10196.

[2] Naveen Kodali u. a. "On Convergence and Stability of GANs". In: *arXiv: Artificial Intelligence* (2018).

[3] *pggan-tensorflow*. https://github.com/henry32144/pggan-tensorflow/blob/master/PGGAN-Tensorflow.ipynb.

[4] Tom O'Malley u. a. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[5] Takeru Miyato und Masanori Koyama. *cGANs with Projection Discriminator*. DOI: 10.48550/ARXIV.1802.05637. URL: https://arxiv.org/abs/1802.05637.

[6] *Flask Dokumentation*. https://flask.palletsprojects.com/en/2.2.x/.

[7] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.