Automating Reading of Ingredient Labels with Computer Vision

Lena Merkli and Sonja Merkli

July 16, 2024

Abstract

With the recent advancements in computer vision and optical character recognition and using a convolutional neural network [11] to cut out the product from a picture, it has now become possible to reliably extract ingredient lists from the back of a product using the Anthropic API [4]. Open-weight or even only on-device optical character recognition lacks the quality to be used in a production environment, although the progress in development is promising. The Anthropic API is also currently not feasible due to the high cost of 1 Swiss Franc per 100 pictures.

The training code and data is available on GitHub: `https://github.com/lenamerkli/ingredient-scanner/`. An inference example can be found on HuggingFace: `https://huggingface.co/lenamerkli/ingredient-scanner`.

This is an entry for the 2024 Swiss AI competition. More information: `https://www.ki-wettbewerb.ch/`.

Contents

Chapter 1

Project Plan

1.1 Architecture

Our computer vision project, Ingredient-scanner, uses multiple layers of artificial intelligence's stacked on top of each other and connected with code. This ensures the computational efficiency as it is possible to aid the individual layers with code and standardize the connections between them.

1.1.1 Image Filters

As the first step, each frame is parsed by image filters which sharpen the edges and reduce the resolution as well as the color spectrum.

1.1.2 First Vision Layer

The first vision layer is a key-point detection convolutional neural network and is going to detect the 4 corners of the packaging. This neural network outputs two additional points at the top and the bottom of the packaging to identify cylindrical objects. We assume as part of our project that spherical products are in-existent.

1.1.3 Image Distortion

All data-points from the first vision layer are used to distort and crop the image in such a way that the back of the product covers the entire rectangular canvas as if it was scanned by a scanner in printer.

1.1.4 Second Vision Layer

Similar to the first one, this is also a key-point detection convolutional neural network. It is going to identify the part of the back of the packaging with the ingredient list. All other image data is discarded, although it can be used in future projects.

1.1.5 Optical Character Recognition

This part turns pixels from an image into text characters. We won't create our own OCR engine as it probably would return sub-optimal results.

1.1.6 Text Parsing with LLM

It is known since the beginning of the existence of OCR that they sometimes return additional white space, leave out white spaces and swap characters. To combat this, the result of the last layer is standardized by a large language model. In order to increase the accuracy, GBNF (Gerganov Bakus-Naur Form) [6] along with a local llama.cpp [8] instance will be in use. This step is also important to distinguish between ingredients and contaminants.

1.1.7 Lookup Table

Information about each ingredient and contaminant will be retrieved from a lookup table. Included in each entry will be at least data on the following parameters: lactose, gluten, vegan, vegetarian, egg, peanut, tree nut, soy and fish. The exact number and contents will be determined once we reached this step. The definition of those will be derived from Swiss law [1].

1.2 Training-Data Aggregation

1.2.1 Video Recording

Short video clips of the back of the product from different angles are recorded to eliminate the hustle to take a lot of photos manually. These videos are cut to remove unusable data if not the entirety of the product has been captured. All frames are extracted from those videos with FFmpeg [20] [13].

1.2.2 Manual Labeling

The pictures resulting from the previous process are labeled manually by determining the corners and the curvature. The results are stored in JSON [18], which is both human- and computer-readable.

1.2.3 Dataset Inflation

In order to save working hours, all already label data points are automatically distorted, rotated and edited in other ways to create new synthetic data. The coordinates of the corners will be edited in the same way.

1.2.4 Hybrid Labeling

Once a first version of the AIs are trained, these can be used to generate the data of not yet labeled pictures. This data will be review and corrected if necessary. We have been inspired by the reinforcement learning from human feedback (RLHF) [15] [23] of large language models.

1.3 Coding

1.3.1 Frame Extraction

Extracts frames from all videos. Used in Video Recording.

1.3.2 Dataset Inflation

Creates new data-points from already existing ones. Used in Dataset Inflation of Training-Data.

1.3.3 Image Sharpening

Sharpens the edges and reduces the resolution as well as the color spectrum. Used in Image Filters.

1.3.4 Image Distortion

Distorts an image based on the corners and the curvature to create the illusion of a flat and rectangular photo. Used in Image Distortion.

1.3.5 Lookup Table

Categorizes the found ingredients according to the database. Used in Lookup Table.

1.4 AI Training

The two AIs will be trained after each other with PyTorch [17] on a local server with a nVidia RTX 4060 with 8GB VRAM.

1.5 Work Distribution

Sonja will be responsible for the aggregation of the training data. Lena does everything else.

Chapter 2

First Vision Layer

2.1 Dataset

The dataset contains just 190 hand-labeled images. Some labels do not or only have parts of the curvature data which can easily interpolated from the other points assuming it lies on a straight line. On GitHub, this dataset is distributed over two directories: /data/full_images/frames and /data/full_images/frames_json.

2.1.1 Data Aggregation

As there was no existing dataset for such a project (hence this paper exists), the authors created a dataset themselves by collecting and labeling photos. A custom viewer, /data/full_images/viewer.py, has aided in the labeling process because it outputs the coordinates of all mouse clicks.

2.1.2 Synthesizing Additional Data

As these 138 data-points are clearly not enough to fine-tune a convolutional neural network, a custom training data synthesizer has been applied. Six different algorithms are in use for this purpose in the following order:

- add_gaussian_noise: Gaussian noise [10] with a mean of zero and a sigma of one is applied to the entire image.
- adjust_brightness: Adjusts the brightness of the image by a random factor between 0.5 and 1.5.
- adjust_contrast: Randomly adjusts the contrast by a value between 0.5 and 1.5.
- $\sim 50\%$ rotate_image: Rotate by 180°.
- $\sim 50\%$ ImageOps.invert: Invert all the values of each color channel.

- $\sim 75\%$ apply_background: Zooms and rotates the image and places it on a random spot on a random background [16].

These are implemented in /data/full_images/generate_synthetic.py. Some of them only have a certain chance to be applied, see the percentages before the function name. All of this results in a 15 to 16 times larger training dataset.

2.1.3 Reinforcement Learning from Human Feedback

Even though it was planned to use RLHF to increase the size of the dataset more efficiently, this idea was abandoned due to the introduction of the aforementioned viewer which already increased productivity enough.

2.1.4 Examples

(a) Significant curvature

(b) Synthetic

Figure 2.1: Example images from the dataset (cropped)

2.2 CNN Architecture

The first vision layer is a convolutional neural network based on the ResNet-18 [11] architecture and weights. This underlying model has been modified to output 12 floating point values and fine-tuned for this project. As an

input, 224*224 images with 3 color channels are used, like in the original ResNet-18.

2.2.1 Loss Functions

A variety of different loss functions were tested for a bit more than 16 epochs each (there is a custom function which determines to exact epoch to stop). Batch size is four and learning rate 10^{-4}. These non-changing settings caused some loss graphs to flatten out very early. Test were conducted on the 28192c4 commit.

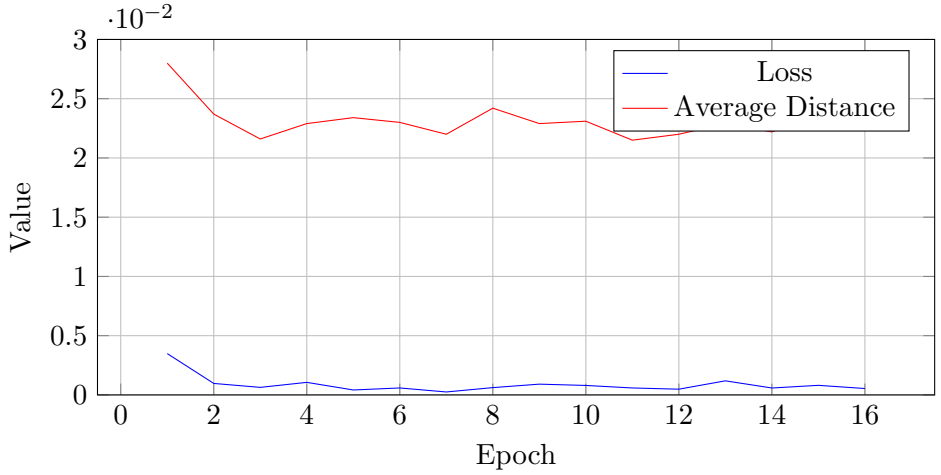

Figure 2.2: MSELoss

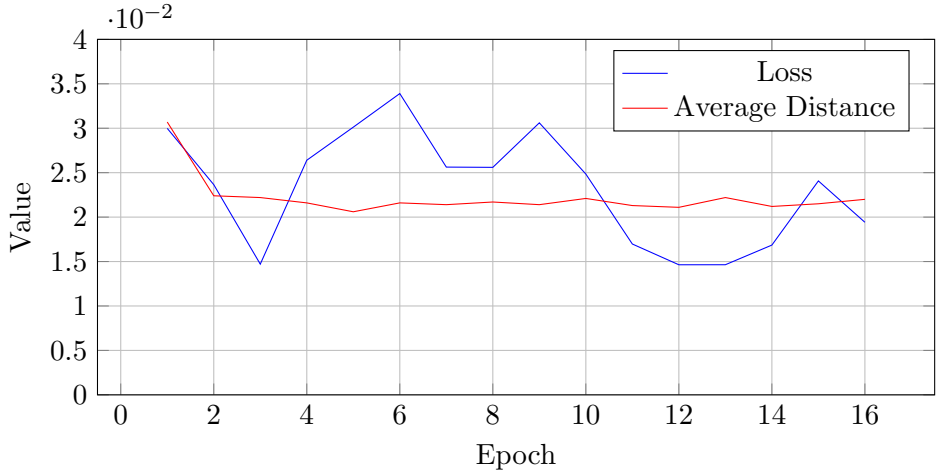

Figure 2.3: L1Loss

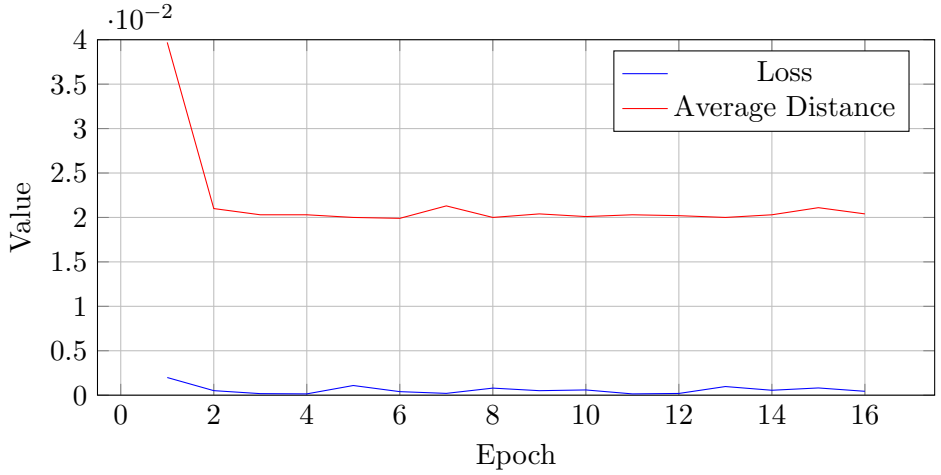

Figure 2.4: SmoothL1Loss

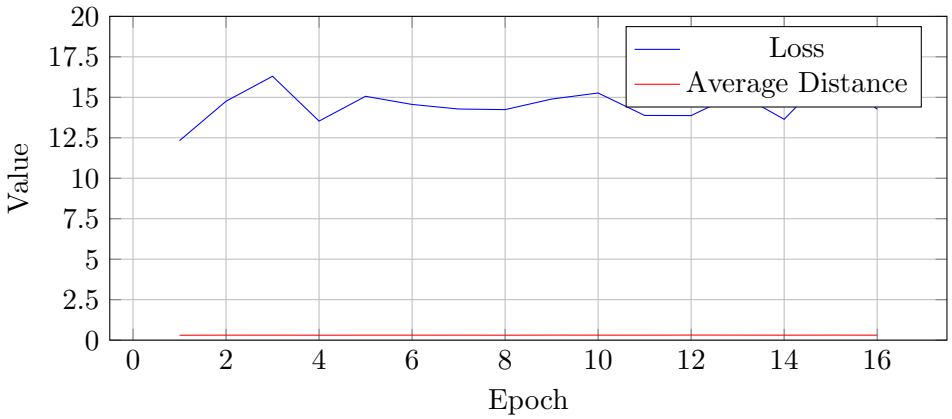

Figure 2.5: CrossEntropyLoss

As it is not visible in the figure for the CrossEntropyLoss, the average distance is about 0.3 ± 0.0125 and does not follow any apparent pattern.

2.2.2 Custom Loss Function

Errors in any two directions in this project are not the same. It is much better to have a too large canvas as opposed to a too small one. A custom loss function has been created to consider this fact. For each point the distance is calculated using Pythagoras theorem. If it lies to the center of the target, the distance is recalculated as $distance = ((olddistance + 1)^{\beta} - 1) * \alpha$.

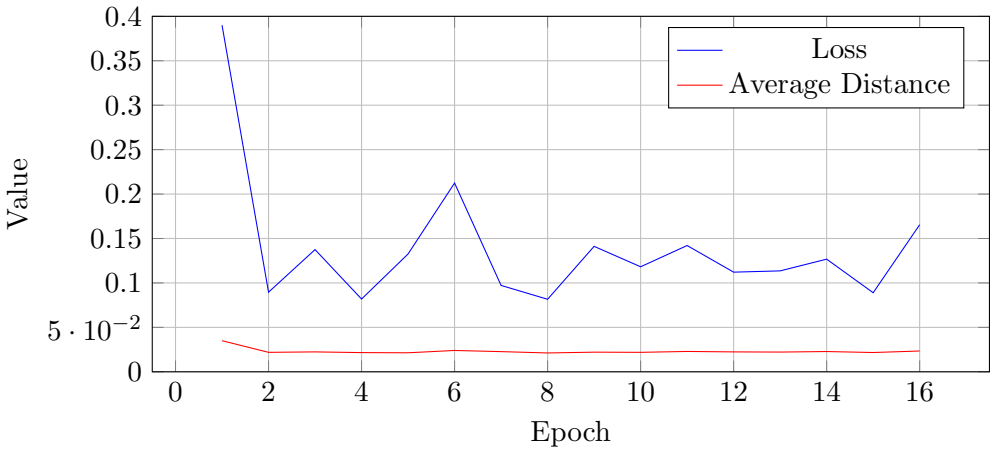

Figure 2.6: IngredientScannerLoss ($\alpha = 1.0, \beta = 1.2$), same parameters as above

2.3 Training

In contrast to the project plan, a RTX 4070 laptop was used. This had the major benefit of faster prototyping and debugging over the planned RTX 4060 server.

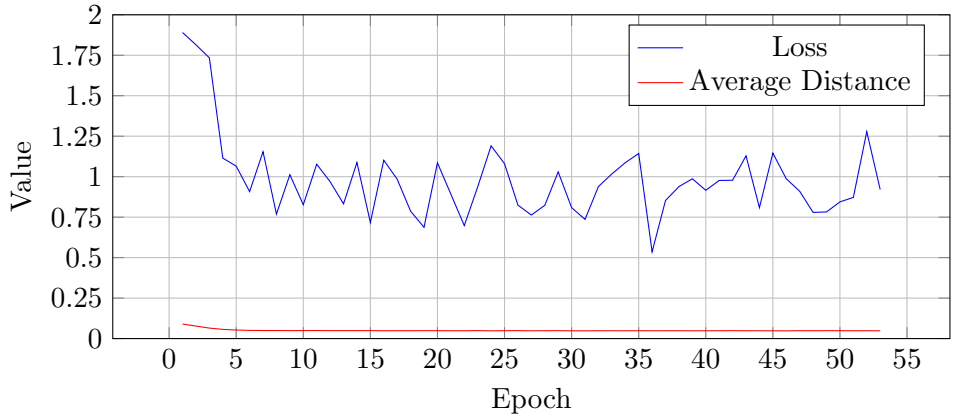

Figure 2.7: Training loss (IngredientScannerLoss, $\alpha = 1.0, \beta = 1.2, batch = 8, learningRate = 5 * 10^{-6}$)

2.3.1 Result

(a)

(b)

Figure 2.8: Images not seen in training (cropped)

Chapter 3

Second Vision Layer

The proposed second vision layer was never developed due to several reasons:

- Optical character recognition is already good enough to recognize the text from the cropped and distorted image from the first layer.
- The authors think after developing the first vision layer that this convolutional neural network would have been unnecessarily hard.
- It would have required to detect text itself as the placement of the ingredients list on the packaging is arbitrary.

Chapter 4

Optical Character Recognition

4.1 Local

Finding an optical character recognition engine with a good enough quality was difficult and a local one even more. Thanks to insider access, a private model was accessible. It is mentioned as "Qwen-VL-Next" in the source code because it has no name yet and the optical character recognition engine is based upon the openly available "Qwen-VL" [2] architecture. Even though a large language model is included, it is not good enough for the purposes of this paper yet. The authors of this papers respect the wish of the provider and owner of it to not release further details.

4.2 API

Because other people do not have access to the local model, a drop-in replacement is available using the Anthropic API [4].

4.3 Image Format

Both the local and online optical character recognition engines have file size limits. Of the tested image file formats, webp was found to be the best one [22] [5].

Chapter 5

Large Language Model

5.1 Dataset

A synthetic data generator created a dataset with 10^4 examples based upon patterns noticed in the optical character recognition engine of the local visual language model. Too many different functions were implemented to fit onto this report, these can be viewed in generate_synthetic.py.

5.2 Training

The local version of unsloth [9] has been used for fine-tuning [21] [14]. The base model is Qwen2-0.5B-Instruct [19] with 4bit bits-and-bytes [3] pulled from `https://huggingface.co/unsloth/Qwen2-0.5B-bnb-4bit`. Training lasted for 256 steps or 0.2 epochs with the adamw_8bit [12] optimizer.

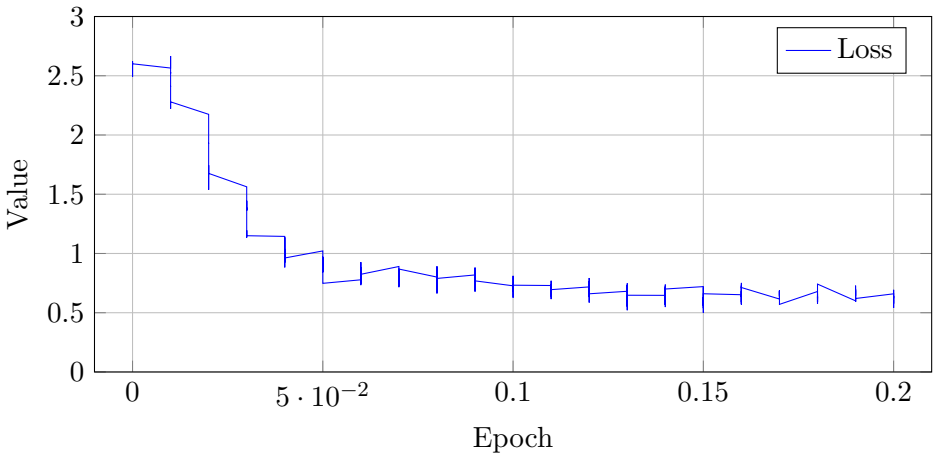

Figure 5.1: training loss (jagged due to rounding errors)

5.3 Compression

The trained model is compressed to Q4_K_M using the standardized GUFF [7] format, version three, for faster inference later on.

Chapter 6

Result & Conclusion

6.1 Examples

(a) Image 1 (b) Image 2

Figure 6.1: Images not seen in training

==========

Zutaten: Cashewkerne, Haselnüsse, Kokosblütenzucker, Kakaobohnen gemahlen, Mandelöl kaltgepresst, Kakaobutter, Himbeeren gefriergetrocknet, Macapulver, Bourbon-Vanille

==========

Ohne Verunreinigungen

==========

Gefundene tierische Zutaten: keine

==========

Gefundene potenziell tierische Zutaten: keine

==========

Gefundene Milchprodukte: keine

==========

Gefundene Gluten: keine

==========

(a) Result for image 1

==========

Zutaten: Datteln getrocknet, Sultaninen geölt, Haselnüsse geröstet, Mandelmus, Cashewnüsse, Haferflocken glutenfrei, Haselnussmus, Mandeln, Dattelsirup, Meersalz

==========

Kann Spuren von Soja, Milch, anderen Nüssen enthalten.

==========

Gefundene tierische Zutaten: keine

==========

Gefundene potenziell tierische Zutaten: keine

==========

Gefundene Milchprodukte: Milch

==========

Gefundene Gluten: keine

==========

(b) Result for image 2

Figure 6.2: Results of the images above in German with OCR over API

==========

Zutaten: Cashewnässe, Haselnüsse, Kakao-Bohnen, Mandelöl, Macapulver, Bourbon-Vanille

==========

Ohne Verunreinigungen

==========

Gefundene tierische Zutaten: keine

==========

Gefundene potenziell tierische Zutaten: keine

==========

Gefundene Milchprodukte: keine

==========

Gefundene Gluten: keine

==========

(a) Result for image 1

==========

Zutaten: Datteln, Sultaninen, Raisins, Sultanines, Haselnüsse, Haumensür, Mandelmus, Cashewenür, Haselnussmus, Mandeln, Dattelsirup, Meersalz

==========

Kann Spuren von Soja, Milch enthalten.

==========

Gefundene tierische Zutaten: keine

==========

Gefundene potenziell tierische Zutaten: keine

==========

Gefundene Milchprodukte: Milch

==========

Gefundene Gluten: keine

==========

(b) Result for image 2

Figure 6.3: Results of the images above in German with local OCR

6.2 Conclusion

Although local (aka. on-device) optical character recognition is not yet ready, we as the authors think of this project as a success. The project is currently not financially feasible due to the Anthropic API costing $\frac{1}{100}$ Swiss Francs. With more advances in machine learning, computer vision, character recognition and compute power, this approach will probably work on the smartphone itself.

A lot of features could be added, such as detecting labels like "lactose-free" or "vegan", reading the identifying bar-code, and allowing multiple camera angles.

The choice of an only 0.5B parameter local large language model was a mistake, as this is apparently not enough for the simple task of converting plain text or markdown into a JSON list.

Due to resource constraints, a lot of synthetic data was used which probably negatively impacts the accuracy of the models. Vegan products are over-represented, as both authors are mostly vegan households. Products with a lot of ingredients were favored in the data collection due to being more interesting.

Bibliography

[1] SR 817.022.16. "Ordonnance du DFI concernant l'information sur les denrées alimentaires". In: *Fedlex* (Feb. 1, 2024).

[2] Jinze Bai et al. *Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond*. 2023. arXiv: 2308.12966 [cs.CV]. URL: https://arxiv.org/abs/2308.12966.

[3] "bitsandbytes". In: *HuggingFace* (Mar. 14, 2024). URL: https://huggingface.co/docs/bitsandbytes/main/en/index (visited on 2024-07-15).

[4] "Claude 3.5 Sonnet". In: *Anthropic* (June 21, 2024). URL: https://www.anthropic.com/news/claude-3-5-sonnet (visited on 2024-07-08).

[5] Benedikt Dornauer and Michael Felderer. *Web Image Formats: Assessment of Their Real-World-Usage and Performance across Popular Web Browsers*. 2023. arXiv: 2310.00788 [cs.PF]. URL: https://arxiv.org/abs/2310.00788.

[6] Georgi Gerganov et al. *GBNF: Formal grammar constrains for llama.cpp*. May 7, 2024. URL: https://github.com/ggerganov/llama.cpp/blob/master/grammars/README.md (visited on 2024-05-07).

[7] Georgi Gerganov et al. *GGUF*. May 24, 2024. URL: https://github.com/ggerganov/ggml/blob/master/docs/gguf.md (visited on 2024-07-15).

[8] Georgi Gerganov et al. *llama.cpp: LLM inference in C/C++*. May 7, 2024. URL: https://github.com/ggerganov/llama.cpp (visited on 2024-05-07).

[9] Daniel Han et al. *unsloth: Finetune Llama 3, Mistral, Phi-3 and Gemma 2-5x faster with 80 percent less memory*. July 15, 2024. URL: https://github.com/unslothai/unsloth (visited on 2024-07-15).

[10] Charles Harris et al. *NumPy Manual: numpy.random.normal*. URL: https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html (visited on 2024-07-05).

[11] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[12] "Introduction: 8-bit optimizers". In: *Huggingface* (Mar. 15, 2024). URL: https://huggingface.co/docs/bitsandbytes/en/optimizers (visited on 2024-07-16).

[13] Karl Kroening et al. *ffmpeg-python: Python bindings for FFmpeg*. July 11, 2022. URL: https://kkroening.github.io/ffmpeg-python/ (visited on 2024-05-19).

[14] Dhanush Kumar. "Qwen2 - Finetuning Qwen2". In: *medium* (June 8, 2024). URL: https://medium.com/@danushidk507/qwen2-finetuning-qwen2-f89c5c9d15da (visited on 2024-06-30).

[15] Nathan Lambert et al. "Illustrating Reinforcement Learning from Human Feedback (RLHF)". In: *Hugging Face Blog* (2022).

[16] Muhammad Ahmad. *Indoor Scenes CVPR 2019*. 2019. URL: https://www.kaggle.com/datasets/itsahmad/indoor-scenes-cvpr-2019 (visited on 2024-06-21).

[17] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[18] Felipe Pezoa et al. "Foundations of JSON schema". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 263–273.

[19] "Qwen2 Technical Report". In: (2024).

[20] Suramya Tomar. "Converting video formats with FFmpeg". In: *Linux Journal* 2006.146 (2006), p. 10.

[21] "Transformers: Fine-tune a pretrained model". In: *HuggingFace* (May 14, 2024). URL: https://huggingface.co/docs/transformers/en/training (visited on 2024-07-15).

[22] James Zern, Pascal Massimino, and Jyrki Alakuijala. *WebP Image Format*. Internet-Draft draft-zern-webp-15. Work in Progress. Internet Engineering Task Force, Apr. 2024. 54 pp. URL: https://datatracker.ietf.org/doc/draft-zern-webp/15/.

[23] Daniel M. Ziegler et al. *Fine-Tuning Language Models from Human Preferences*. 2020. arXiv: 1909.08593 [cs.CL].