

"That's Not What I Ordered!" - A Deep Learning Approach to Beer Classification

Barry Shafran and Lavie Lederman

Faculty of Electrical and Computer Engineering, Technion - Israel Institute of Technology

August 13, 2025

1 Abstract

This project explores deep learning based beer classification from images, motivated by the challenge of identifying beer types visually, whether to confirm an order in a restaurant or for automated beer recognition. We tested state of the art models, including custom CNNs, DINO, and EfficientNet, and compared their performance. Data collection was achieved using a custom pipeline leveraging the Untappd beer website for labeled images and YOLOv8 for object detection and cropping of relevant beer images. The study also included a GUI based 'human benchmark' game for comparison.

2 Introduction

Beer classification from images is a non-trivial computer vision task. The project aims to automatically identify beer types, such as cider, wheat, lager, IPA, stout, and not beer. using visual cues from user submitted photos. These photos often contain noise, irrelevant backgrounds, or different presentation styles, making classification challenging. The project combines dataset creation, preprocessing, model training, and comparative evaluation.

2.1 Problem statement

2.1.1 Absence of Standardized Datasets

There is no publicly available, labeled dataset for this task. We therefore constructed a custom dataset using publicly sourced images and YOLOv8 [2] model for the object detection, bounding box and cropping out task.

2.1.2 Visual Similarity Between Classes

Several beer styles exhibit overlapping visual characteristics in terms of color, foam transparency, and glassware, leading to confusion between classes. During testing the GUI we built to determine a human benchmark for this project, out of 6 classes human seemed to only be able to consistently classify the "stout" class (due to its dark and unique color compared to other beers) and the "not beer" control group. We assume the model will face similar difficulties. This issue is exemplified in Figure 1:



Figure 1 – An example of 4 different beer types, 3 of which have very similar attributes

2.1.3 Performance Degradation in Non-Ideal Conditions

Variations in lighting, camera motion, and background clutter can significantly reduce classification accuracy in practical applications.

In our case, the dataset was collected from user submitted photos, which naturally include these imperfections. Images vary widely in brightness, sharpness, perspective, and background content, often containing distracting objects or partial occlusions of the beer glass.

Because of this, our models must be robust to such non ideal conditions, learning to focus on beer specific features while ignoring irrelevant background noise. Successfully handling these challenges is essential for deploying the system in practical scenarios where image quality cannot be controlled.

3 Data Acquisition & Preprocessing

3.1 Collection

Due to the unconventional task, no publicly available dataset existed for this mission. Therefore, we implemented a custom two stage data collection pipeline, sourcing images from Untappd [1], a popular consumer beer review platform, and organizing them into clearly labeled categories for training, based on the categories in the Untappd website. To access the links, we edited a cookies JSON file with our login data (this file, of course, is not included in the repository).

In the first stage, links for the photos were collected using Selenium [5] (to generate the html and load photos dynamically via JavaScript) and BeautifulSoup [6] (To extract a list of .jpg URLs to a CSV file). Searches were performed using the labels (“IPA”, “Stout”, “Wheat”, “Lager”, and “Cider”) and grouped accordingly. We limited the link collection for 300 per beer brand and 1000 pictures per label, manageable size while maintaining diversity across brands.

In the second stage, due to problems with downloading directly all the data, we used separate Python script to read the CSV file and download the photos to a specific directory, while keeping track of already downloaded photos and ignoring them in later runs of the same script. The images were saved into the classes' directories (the labels).

3.2 Data sorting

In the next stage, and in order to sort out all unrelated pictures and backgrounds we used YOLOv8 and its predefined labels: ['cup', 'wine glass', 'bottle']. We extracted only the item recognized as 'cup' and 'wine glass' class for this project, manually deleting unrecognizable items such as red plastic cups, beer cans, and glasses that did not appear to contain beer (e.g., juice or water). The deleted images were used to bring the "control" class of "not beer" to address the issue of false positives. An example for this process can be seen in Figure 2:



Figure 2 – the classes collected by YOLOv8. Only the classes, 'wine glass' and 'cup' were relevant

All the collected bounding boxes were collected if their confidence exceeded 50%. After this process all the bounding boxes were sorted into labels, and the removed photos were labeled as "not beer".

3.3 Augmentations

After splitting the data to train, validation and test sets (as will be explained in section [5]) we applied a set of image transformations to improve robustness and generalization. For the **training set**, we used augmentations to simulate realistic variations in lighting, orientation, and framing. For the **validation set** we applied only resizing and normalization, and for the **test set** minimal transforms were applied – only resizing and conversion to tensor.

In this stage we started training the model, but we encountered low results, even after 50 epochs (67-72%) and the model started to show signs of overfitting as can be seen in Figure 3:

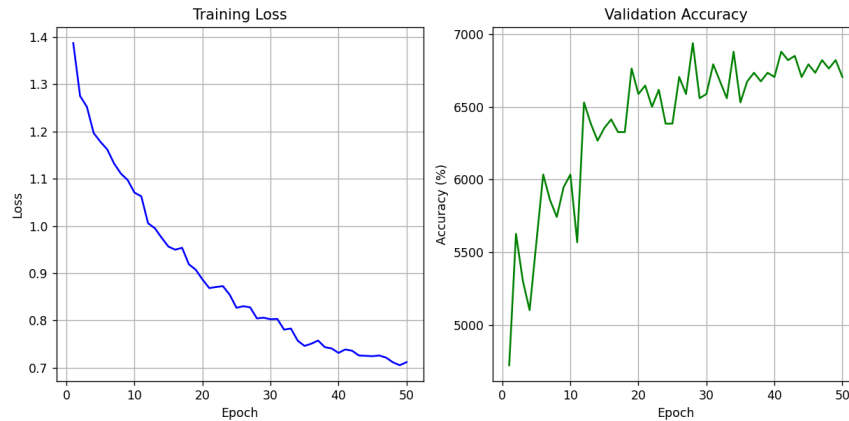


Figure 3 – validation set accuracy and training loss as function of epochs. The maximum accuracy reached was 72%, with clear indications of overfitting.

The dataset consisted of 2,744 training samples, 343 validation samples, and 343 test samples. Increasing the model size was not preferred due to the limited dataset. Instead, we chose to increase generalization by generating one additional augmented sample for each existing training image.

The chosen augmentations were resizing, random crop, random horizontal flip and rotation, random affine and random perspective. This process doubled the size of the dataset, and increased the model performance– as can be seen in section [X].

3.4 Human Benchmarking via Beer Classifier Test GUI

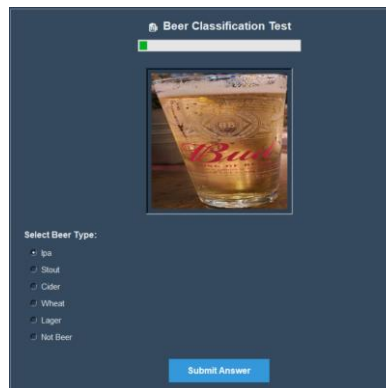


Figure 4 – Example from the GUI built to determine a human benchmark

To establish a baseline for the task, we developed a custom Beer Classifier Test GUI, as can be seen in Figure 4. This application tested participants with a random set of images from the dataset and graded their answers. The average accuracy achieved by the participants was approximately 62%, providing a useful reference point for evaluating model performance.

It is important to note that this benchmark was somewhat inflated due to some factors that made certain classes particularly easy to identify:

- Highly distinctive visual cues — For example, *stouts* were typically very dark in color, making them easily distinguishable from other beer types. Similarly, the Not Beer category functioned as a control class, often containing objects or beverages with no resemblance to beer, resulting in straightforward classification for human participants.
- hints in images — In some cases, the images themselves contained labels, such as a beer glass with “Cider” or “IPA” printed on it. While this mirrors potential real-world cues, it does not reflect purely visual reasoning and therefore inflates the human benchmark relative to a purely visual classification task.

4 Architecture

We chose simple architecture for our model, as can be seen in Figure [5]:

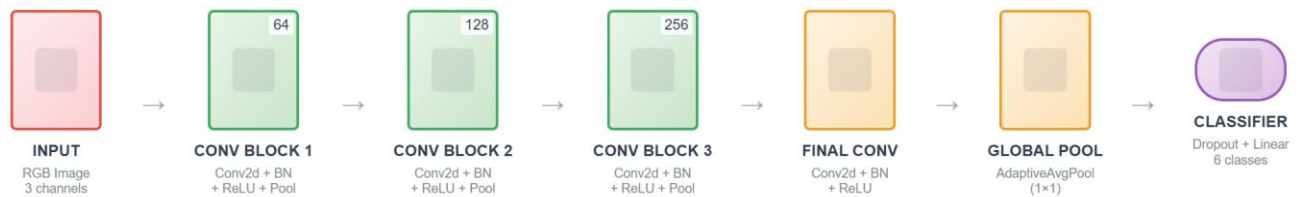


Figure 5 – The Architecture, our convolutional blocks followed by a pooling layer, and a classifier consisting of dropout and a linear layer for classification into six classes.

The model was intentionally kept small and simple, as seen in Figure 5, due to the limited size of the dataset, reducing the risk of overfitting. The classifier architecture, batch size, and normalization layers were selected from a variety of candidate models to best suit our objectives and were generally based on the implementation from Homework 2 of our course. The model consists of total of 1,555,014 parameters.

We compared our model to two state-of-the-art image classification models:

EfficientNet[4], a family of CNNs that scales depth, width, and resolution using compound scaling to minimize parameters, and **DINO**[3], a self-supervised vision transformer that learns high-quality image representations without labeled data, enabling strong performance on downstream tasks. Both were trained using three different strategies: freeze-out, partial training, and full model training and were compared to each other and our model.

Random search was performed to choose the hyper parameters including: Learning Rate, patience, weight decay – tested on validation.

5 Results

5.1 Basic CNN architectures

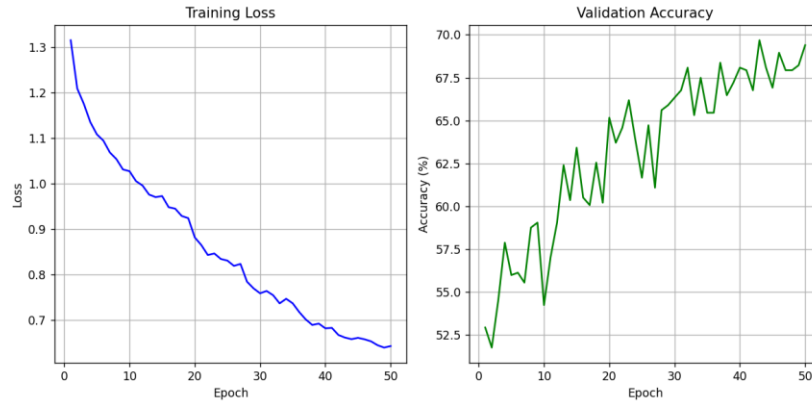


Figure 6 – Basic CNN models result, Test accuracy is 72.59%, while not yet plateaued, we can see sign of beginning to overfit towards final epochs.

Our custom CNN architecture was designed as a low cost, low complexity alternative to large pre trained networks. The goal was not to match the absolute state of the art but to produce a classifier that could be trained efficiently from scratch while still outperforming a human level benchmark for this task. In this regard, the model succeeded: despite its simplicity and limited parameter count, we achieved performance comfortably above the measured human baseline.

However, compared to transfer learning approaches, the basic CNN model's accuracy plateaued relatively early and did not exceed ~73% while testing. This performance gap underscores the inherent limitations of small models trained from scratch when faced with a complex visual classification problem. Nonetheless, these models represent a cheap and easy alternatives that trade accuracy for efficiency.

5.2 State of the art architectures

The state-of-the-art models were trained for 20 epochs as can be seen in Figure 7:

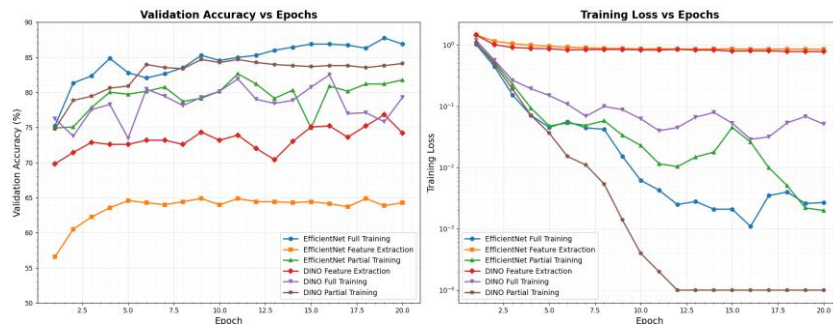


Figure 7 – Validation Accuracy and Training Loss for EfficientNet and DINO under three training strategies over 20 epochs. We can see that DINO partial training achieved the fastest convergence and lowest loss. Feature extraction approaches minimal improvement.

The state-of-the-art models achieved results which were an improvement upon what our models could achieve, (excluding feature extract only models) as can be seen in Table 1 . Significantly EfficientNet Full fine-tuning reaching 87.76% the highest test result overall. Interestingly, EfficientNet Partial training (83.67%) and DINO Partial training (83.53%) performed only slightly worse than their fully fine-tuned counterparts, despite training fewer parameters. In fact, DINO Partial (83.53%) outperformed DINO Full (80.76%), suggesting that freezing parts of a powerful pre-trained backbone can help preserve useful generic representations and act as a form of regularization.

By contrast, pure feature extraction approaches (where the pre-trained network is used as a fixed feature generator) performed worse (EfficientNet scored 67.78%, while DINO scored 73.91%), indicating that partial training might be well worth the extra training required, and might be worth its own project to test the amount of layers retrained as another hyper parameter. In addition, it can be seen in Figure 7 that these models stopped learning at a fairly early stage (after 3-4 epochs) and got stuck, as expected.

In summary, transfer learning with partial fine tuning makes an excellent balance between computational efficiency and accuracy. While full fine tuning of EfficientNet scored the highest test score in our project, The fact that DINO partial model outperformed DINO full model is a more interesting result we achieved, pointing to the possibility of matching and even improving upon the results of fully retraining the state-of-the-art models with less computing required.

	Basic CNN	EffNet full train	EffNet freeze out	EffNet partial	Dino Freezout	Dino full	Dino partial
Score[%]	72.59	87.76	67.78	83.67	73.91	80.76	83.53

Table 1 – Test accuracy comparison of all methods. As shown, state-of-the-art models outperformed basic CNN, even when trained for fewer epochs.

6 Conclusion

For this project, we faced a classification problem with no existing labelled dataset. To address this, we combined web scraping from Untappd with YOLOv8 object detection to automatically crop and label relevant images. This demonstrated how existing models can be repurposed to efficiently generate labelled datasets for niche problems.

Our own lightweight CNN architecture was designed to be simple, fast, and inexpensive to train. Despite the small size, the model achieved 72–73% accuracy, comfortably above the ~62% human benchmark, making them a strong practical option when resources are limited or fast deployment is required.

State-of-the-art models, particularly EfficientNet and DINO, achieved higher performance, with EfficientNet full fine-tuning reaching 87.78%. Notably, DINO partial fine-tuning outperformed its fully trained counterpart, suggesting that freezing parts of a powerful backbone can preserve valuable generic features, improve accuracy, and reduce computational cost. These findings position partial fine-tuning as a promising direction for future work.

We can see a final comparison of the models on a randomly chosen photo in Figure 9:

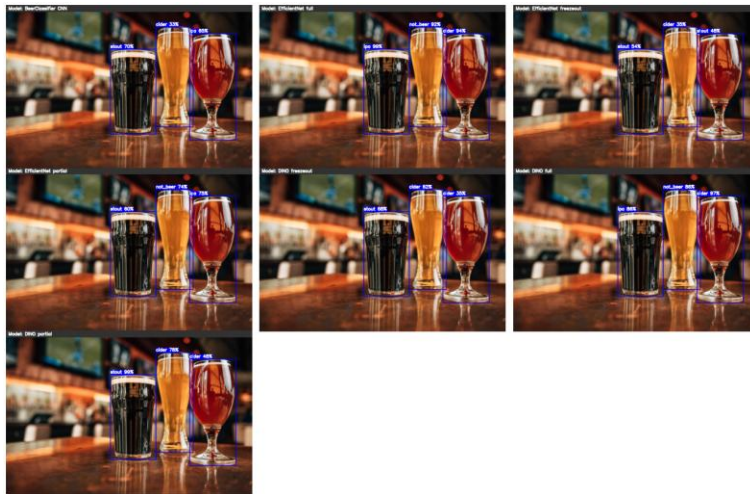


Figure 9 – Resulted labels on 3 beers (random internet picture) – while most of the models label stout correctly, they do have trouble labeling the other beers.

7 Ethics Statement

Lavie Lederman & Barry Shafran

“Thats not what I ordered!” - A Deep Learning beer classification model

This project develops a deep learning system to classify beer types from photographs. It aims to automate beer recognition for applications like confirming customer orders, cataloging drinks, or enhancing user experiences, even when images are noisy, taken from varied angles, or contain background clutter.

7.1 LLM-Generated Responses

We asked ChatGPT 5 for the given prompts, and received the following answers

Prompt:” List 3 types of stakeholders that will be affected by the project.”

- **Restaurant and Bar Owners** – Can use the system to ensure correct drink delivery and improve customer satisfaction.
- **Customers/Beer Enthusiasts** – Benefit from quick verification of beer type, reducing order mistakes and enhancing the experience.

- **Technology Developers and Researchers** – Gain insights from the model’s design and performance for similar visual classification tasks.

Prompt:” What will an explanation that is given to each stakeholder look like? “

- **Restaurant and Bar Owners** – “Our beer classification system uses advanced image recognition to verify the type of beer served in real time, helping reduce order mistakes, improve service efficiency, and enhance customer trust.”
- **Customers/Beer Enthusiasts** – “This tool lets you snap a photo of your drink and instantly confirm its type, ensuring you get exactly what you ordered and adding a fun, interactive way to explore different beers.”
- **Technology Developers and Researchers** – “The project demonstrates a complete pipeline from dataset creation with YOLO-assisted cropping to training and evaluating multiple deep learning models (CNN, EfficientNet, DINO), providing a practical case study in applying state-of-the-art visual classification to a niche domain.”

Prompt:” Who is responsible for giving the explanation to each stakeholder? “

- **Restaurant and Bar Owners** – The **project manager** or a **business development representative**, focusing on operational benefits and return on investment.
- **Customers/Beer Enthusiasts** – The **marketing or community engagement team**, highlighting ease of use, accuracy, and the fun aspect of the system.
- **Technology Developers and Researchers** – The **technical lead** or **research team members**, explaining the architecture, dataset creation process, and evaluation results in detail.

7.2 Reflection on LLM Responses

We think the generative AI’s answer need to changed in such a way to provide a more balanced answer and try to replicate a more critical way of thinking, while its answers were correct in regard to our project, the model overlooks possible downsides and issues with the project, such as limitations of our model and possible mistakes. Furthermore the model does not question the impact a project will have, its seems to assume it is(due to the prompt) and reason its way backwards from there.

8 References

- [1] Untappd, “Discover Beer,” 2024. [Online]. Available: <https://untappd.com>
- [2] Ultralytics, “YOLOv8 Models Documentation,” 2024. [Online]. Available: <https://docs.ultralytics.com/models/yolov8/>

- [3] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging Properties in Self-Supervised Vision Transformers," Facebook AI Research, 2021. [Online]. Available: <https://github.com/facebookresearch/dino>
- [4] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019. [Online]. Available: <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>
- [5] Selenium, "Selenium with Python — WebDriver, Waits, and Expected Conditions," 2024. [Online]. Available: <https://selenium.dev/documentation/>
- [6] L. Richardson, "Beautiful Soup Documentation," 2024. [Online]. Available: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>