TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

Henkel AG & Co. KGaA

Final report of project:

Deep Learning for Detergents

Authors	B.Sc. Fatma **Dirman**
	B.Sc. Mahmoud **Hassan**
	B.Sc. Baris **Onarici**
	B.Sc. Guangyao **Quan**
	B.Sc. Mikhail **Tereshkin**
Mentor(s)	Andrii **Kleshchonok** (Henkel)
	Florian **Roscheck** (Henkel)
Co-Mentor	M.Sc. Cristina Cipriani
Project Lead	Dr. Ricardo Acevedo Cabra (MDSI)
Supervisor	Prof. Dr. Massimo Fornasier (MDSI)

Jul 2023

Abstract

This project focuses on utilizing deep learning techniques to develop detergents at Henkel. The specific objective is to expedite the assessment of aesthetic degradations in liquid detergents during prolonged exposure to various climate conditions (known as shelf-life testing). To achieve this, an automated system is being created that can cluster and predict the behavior of detergents during large-scale storage tests using image analysis. The project dataset consists of over 8,000 high-quality storage test images, each labeled by human experts and accompanied by relevant metadata. Preprocessing techniques have been applied to optimize the dataset for recognition and segmentation tasks. Subsequently, image classification and segmentation techniques are employed to analyze the data.

Two separate image classification pipelines has been implemented, encompassing both binary and multiple classifications. These pipelines leverage traditional Convolutional Neural Networks (CNNs) as baseline along with the cutting-edge Contrastive Language-Image Pretraining (CLIP) model. Once the pipelines have undergone training and evaluation, they are registered and deployed to the Azure workspace. Optionally, an endpoint can be set up to facilitate batch predictions. This allows users to submit batches of images for classification and receive the corresponding scoring results and confidence scores in return. By deploying the pipelines and providing batch prediction capabilities, the system enables efficient and user-friendly utilization of the trained models.

The next goal is to extract relevant features from images to be utilized in future tasks such as time series predictions. This was done by first segmenting the images into distinct regions, followed by extracting features from each segment. Four distinct methods for image segmentation have been utilized: SAM, CAM/Grad-CAM, OpenCV and Edge Detection. After careful evaluation, the Edge Detection method implemented with the OpenCV package is selected due to its simplicity and efficiency, which align with the project's specific requirements.

Once the images are segmented, the next step involves analyzing the segmented images by extracting predefined features and encapsulating them into feature vectors. This process is known as featurization and characterization. The chosen features are human-interpretable and provide valuable insights. To address any incorrect label assignments and streamline future labeling efforts, a feature remapping system is developed. This system employs clustering and classification techniques to ensure accurate and consistent labeling. By utilizing these techniques, the workflow aims to improve the overall quality of the labeling process and enhance the reliability of the collected data.

Contents

Abstract **1**

1 Introduction **3**
1.1 Problem Definition . . . 3
1.2 Project Goal . . . 3

2 Statistical Data Exploration **4**
2.1 Dataset Features . . . 4
2.2 Preprocessing . . . 5

3 Use Cases: Methods and Results **6**
3.1 Baseline Binary Classification Pipeline . . . 6
3.1.1 Modeling . . . 6
3.1.2 Results . . . 7
3.2 CLIP Pipeline . . . 8
3.2.1 Modeling . . . 8
3.2.2 Results . . . 10
3.3 Multi-class Classification Pipeline . . . 11
3.3.1 Modeling . . . 11
3.3.2 Results . . . 12
3.4 Image Segmentation . . . 13
3.4.1 Segment Anything Model . . . 13
3.4.2 CAM/Grad-CAM . . . 16
3.4.3 OpenCV . . . 17
3.4.4 Edge Detection . . . 18
3.5 Featurization . . . 19
3.5.1 Featurizers . . . 19
3.5.2 Characterizer . . . 22
3.6 Feature Remapping . . . 22
3.6.1 K-Means Clustering . . . 22
3.6.2 t-Distributed Stochastic Neighbor Embedding (t-SNE) . . . 22
3.7 Uniform Manifold Approximation and Projection . . . 23

4 Discussions **24**
4.1 Future Works . . . 24

5 Conclusions **25**

Appendix **27**

1 Introduction

1.1 Problem Definition

The development of detergents at Henkel requires efficient methods for the assessment of aesthetic degradations in liquid detergents during prolonged exposure to various climate conditions. This process is also known as shelf-life testing. The manual evaluation of thousands of storage test images is time-consuming and prone to human error. Also, subjective interpretation of manual tests leads to varying quality of data annotation. Therefore, there is a need to expedite the detergent assessment process through automated image analysis techniques.

1.2 Project Goal

The primary goal of this project is to offer Henkel experts a customized machine learning pipeline that is readily available for use. The focus is on developing an automated image labeling system that can be effortlessly implemented. By providing a comprehensive pipeline, the aim is to enable Henkel to optimize the detergent shelf-life testing process, minimize manual workload, and improve the precision of decision-making.

It is intended to create a versatile classification pipeline capable of accurately labeling detergents, both in binary and multiclass scenarios. Leveraging image analysis and machine learning techniques, the objective is to build a robust and efficient system that can assess images taken during shelf-life testing both qualitatively (*good*/*bad*) or assign trained aesthetic failure modes (e.g. cloudy, sedimented).

Furthermore, the importance of interpretablity in the solution is recognized, particularly for the lab experts. It is understood that lab experts need to comprehend and trust the underlying processes and decisions made by the system. Therefore, the commitment is to ensure that the solution is highly interpretable, providing clear explanations and insights into the reasoning behind its outcomes.

Interpretability is intended to be achieved by segmenting the images into distinct regions, providing an intuitive understanding of the specific attributes and variations within the detergent samples. Additionally, the extraction of predefined features from these segmented regions will enable human-interpretable and informative representations of the image data. Another critical objective involves extracting relevant features from the images to establish the foundation for future tasks, such as time series predictions.

In summary, the project aims to streamline and expedite the assessment of aesthetic degradations in liquid detergents during shelf-life testing by developing an automated system that incorporates image classification, segmentation, feature extraction, and feature remapping. By automating the classification process and providing a ready-to-use machine learning pipeline, a reliable, easy-to-understand, reusable and retrainable, objective, and efficient solution is aimed to be offered to Henkel experts for automated image labeling.

2 Statistical Data Exploration

2.1 Dataset Features

The dataset utilized in this project has been provided by Henkel, comprising over 4,000 pairs of consistent and high-quality images of liquid detergent samples stored under various climate conditions. Each image is accompanied by its corresponding human-assigned labels and metadata, resulting in a total of 8,861 images, distributed across 12 distinct classes. These classes were assigned manually by the experts in Henkel based on the aesthetic degradation observed in each detergent sample. An image pair consists of a projected side-view image (originally labeled as *Abwicklung*) and a top-view image (originally labeled as *Boden*), as depicted in Figure 2. These labels are assigned based on the angle at which each image was captured.

However, there are some challenges within the dataset that need to be addressed. Firstly, there is a class imbalance issue, such as the presence of approximately 250 multi-phase images out of the total 8861 images. Figure 1 provides an overview of the class distribution among the side view images, highlighting the inherent imbalance in the dataset. It can be seen that there are only 125 multi-phase images (2-phase and 3-phase images are considered to be multi-phase), while more than 1600 clear images are present.

Since the multi-class classification pipeline, which was developed for this project, splits the images into three classes, namely clear, multi-phase, and others, this imbalance can significantly impact its performance. This variation in the number of instances across different classes underscores the need to address the class imbalance challenge in subsequent analysis and modeling tasks.

It is important to note that the class imbalance observed in the dataset is to be expected. As this dataset serves as a confirmatory test during the development of detergents, it is natural to anticipate that only a fraction of the samples would exhibit aesthetic degradation. The dataset reflects real-world conditions, where only a limited number of liquid detergent samples are likely to demonstrate such degradation.

Furthermore, the dataset contains some images with incorrect labels, particularly concerning the angle at which they were captured. For instance, certain images were categorized as side-view images but actually featured a top view. This labeling discrepancy needs to be rectified to ensure the accuracy and reliability of the automated image analysis techniques.

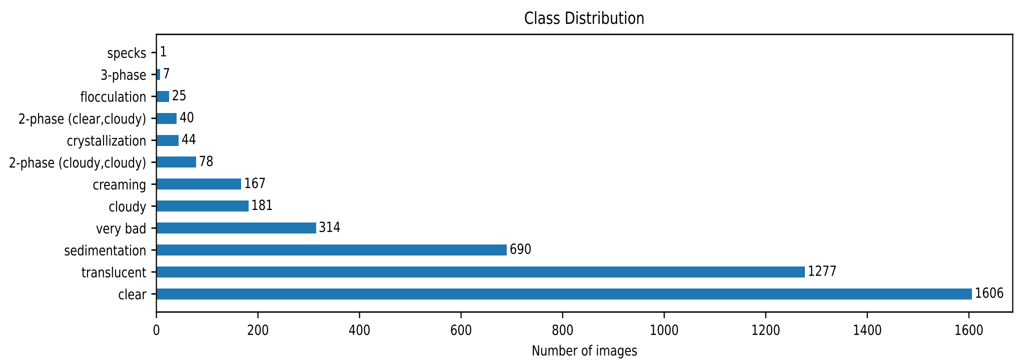


Figure 1: Class distribution of the projected side view images in total of 4430

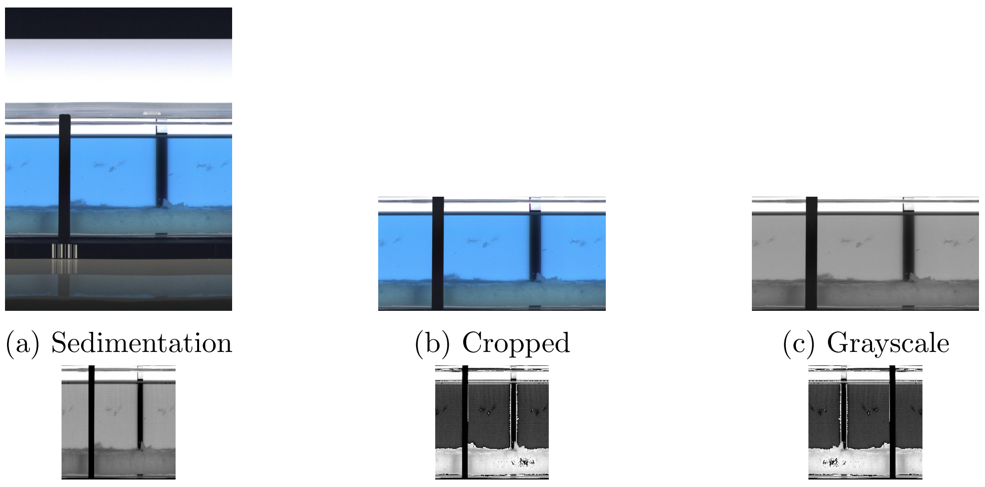
(a) Side view (b) Top view

Figure 2: *Sedimentation* image pair

2.2 Preprocessing

Data preprocessing involves several steps to prepare the images for subsequent analysis and machine learning tasks. As a first step, cropping allows for the removal of unwanted, irrelevant image parts and focusing on specific regions of interest. For instance, the black and white areas at the top of the image, and the reflected part at the bottom of the image are removed, so that only the image area remains, where the stored liquid detergent and its features can be seen. Figure 3b displays the remaining image part. Position of this area to be cropped is consistent within the entire dataset.

Grayscale conversion simplifies the data representation, while resizing ensures uniformity by bringing all images to a standardized dimension based on the model input shape. In the next step, applying a mask for the pixel intensity values below a threshold helps isolate the irrelevant areas in the images, such as the black bars in the detergent tank, which can not be easily removed from the image by cropping. Image normalization step is applied on the dataset using the mean and standard deviation values of training set. Lastly, data augmentation techniques, for instance flipping, are utilized to increase the size and diversity of the training dataset. Figure 3 shows the data transformation steps applied on a sedimentation image. These steps are optional and can be configured for the specific pipeline requirements.


(a) Sedimentation (b) Cropped (c) Grayscale

(d) Resized and Masked (e) Normalized (f) Data Augmentation

Figure 3: Data transformation steps

3 Use Cases: Methods and Results

3.1 Baseline Binary Classification Pipeline

A re-trainable and fully automated system was designed to train our models to detect whether the liquid is considered *bad* or not, where a detergent instance is considered bad if it displays sedimentation, clotting, or any other similar phenomena. In our dataset, the *clear* class is considered as *good*, while the other classes are labeled as *bad*.

This comprehensive pipeline consists of multiple interconnected components, each serving a unique purpose to ensure optimal performance and seamless deployment.

3.1.1 Modeling

The baseline model mainly employs Convolutional Neural Network[7], which has demonstrated exceptional performance in image-based tasks due to their ability to automatically learn hierarchical representations from the data. The model mainly consists of a sequence of convolutional, max-pooling and batch-normalization layers followed by fully-connected network, Additionally, dropout regularization is applied after selected layers, to prevent over-fitting and improve the model's ability to generalize to unseen data.

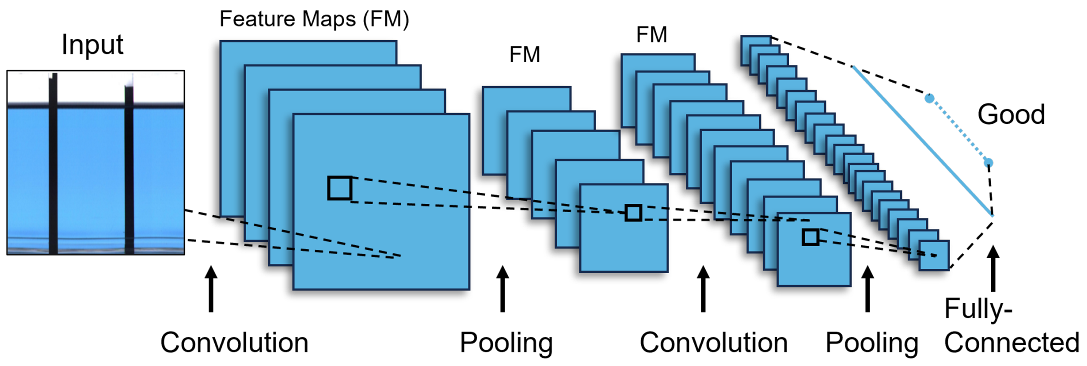


Figure 4: Convolutional Binary Classification Example

Pipeline Components

- **Data preparation**

  The component first iterates over the data from the input directory and extracts the class names and relative information that could exists in the filenames, afterwards it assigns the labels based on the task (Binary/Multi-class) and the input *good* classes. Finally the component splits the dataset in train,test and validation sets based on the input parameters.

  To ensure a representative and balanced distribution of *good* and *bad* classes in both the training and validation sets, a stratified split technique was employed, which maintains the same class distribution in each subset as that of the original dataset.

- **Training**

  Firstly, the component takes as input the data from the previous stage, which includes the pre-processed dataset. Optionally, an input model can also be provided to the training component. This allows for the possibility of starting the training process from a pre-trained model.

  During the training phase, the Binary Cross Entropy loss is used as the objective function. Once the training and validation cycles are completed, the trained model, along with its optimized parameters, is passed to the subsequent components in the pipeline for further deployment and evaluation.

- **Scoring and Evaluation**

  The scoring component takes the trained model and validation/test sets as input, generating predicted probabilities and labels for each sample. These outputs are then passed to the evaluation component, which calculates metrics such as accuracy, precision, recall, and F1-score. Additionally, the evaluation component analyzes the performance separately for each image type, providing insights into the model's classification capabilities. Visualizations such as the confusion matrix are also saved with the experiment run and can be easily downloaded or viewed at any time, facilitating the interpretation and analysis of the model's performance.

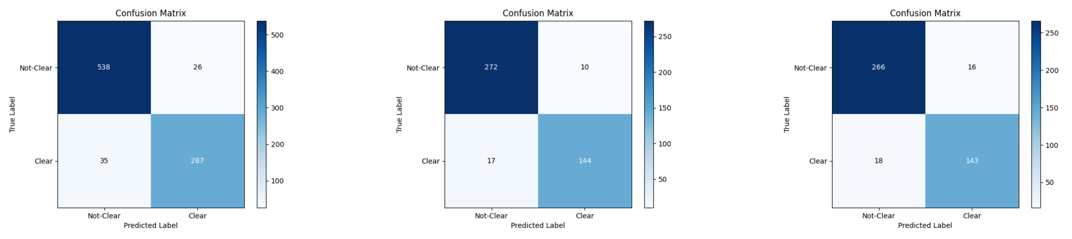
- **Deployment**

  The component begins by registering the trained model in the AzureML[1] workspace, making it easily accessible for future use. This step ensures that the model is properly tracked and versioned, facilitating re-producibility and model management. Next, the model is deployed as a batch-endpoint, leveraging a compute cluster to handle incoming requests efficiently. This endpoint exposes a custom invocation script that receives input images to be classified and generates predicted labels and provides a measure of certainty or confidence associated with each prediction.

3.1.2 Results

In this section, we present the results and performance evaluation of the Baseline binary classification model. We first analyze the confusion matrices to assess the model's predictions and actual labels. Three confusion matrices are provided: one for the overall performance and separate matrices for each image type as seen in Figure 5. It is important to highlight that during the evaluation, the *clear* class was considered as *good*, while the other classes were labeled as *bad*. This categorization aligns with the specific requirements of our problem statement and allows for a thorough assessment of the model's ability to accurately identify *good* instances.

[1]`https://azure.microsoft.com/en-us/products/machine-learning`, [accessed 20.07.23]



(a) Total (b) Side View (c) Top View

Figure 5: The confusion matrices of the baseline Binary classification model

The model achieved impressive performance with an overall accuracy of **0.94**, precision of **0.92**, and recall of **0.89**, effectively distinguishing between *good* and *bad* instances regardless of the image type. Additionally, we examined the precision and recall scores for each image type as shown in table 1.

Image type	Accuracy	Precision	Recall
All views	0.94	0.92	0.89
Side View	0.94	0.93	0.9
Top View	0.93	0.95	0.85

Table 1: Precision and recall scores for each image type

3.2 CLIP Pipeline

3.2.1 Modeling

CLIP (Contrastive Language-Image Pre-Training)[9] is a remarkable architecture that has been pre-trained using diverse pairs of images and corresponding text descriptions. Unlike traditional approaches, CLIP is designed to predict the most relevant text snippet given an image using natural language instructions, without directly optimizing for the specific task. This unique capability is analogous to the zero-shot capabilities demonstrated by models like GPT-2[8] and GPT-3[1]. The emergence of CLIP marks a major advancement in the field of computer vision, revolutionizing the way we approach image understanding and analysis.

Therefore, transfer learning is pivotal when integrating CLIP into our classification pipeline, especially given the complex features of our detergents dataset, including stratification, turbidity, and crystallization. CLIP's ability to recognize hidden visual patterns makes it an excellent fit for this application. By doing so, we can harness its acquired image and text representation capabilities during pre-training phase for our specific classification task. The utilization of CLIP typically follows a concise three-stage procedure, as illustrated in Figure 6. These stages encompass contrastive pre-training as the initial step, followed by the creation of a dataset classifier using label text, and culminating in its application for zero-shot prediction in the final step.

Pre-training

In order to effectively utilize this model for our classification task, it must undergo a pre-training phase as mentioned earlier, which is distinct from the training process within

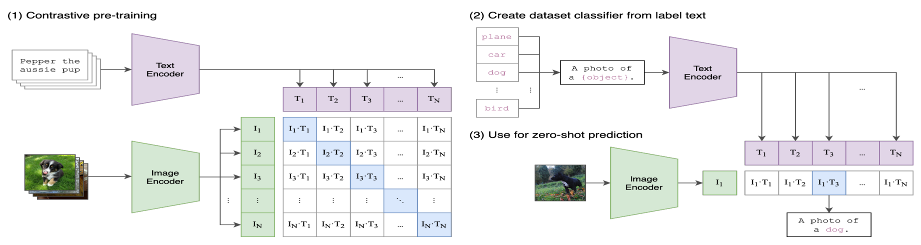

(1) Contrastive pre-training
Pepper the aussie pup
Text Encoder
Image Encoder
(2) Create dataset classifier from label text
plane
car
dog
bird
A photo of a {object}.
(3) Use for zero-shot prediction
A photo of a dog.

Figure 6: A new training method: Contrastive Language-Image Pre-training (CLIP)[9]

our pipeline. This pre-training process involves contrastive learning, vision-language pre-training, and multi-modal fine-tuning. Through these steps, the CLIP model achieves impressive performance across a range of visual tasks. Once the pre-training is complete, we can obtain a model checkpoint directly through the CLIP API. To serve our specific needs, we will further enhance its capabilities through the introduction of linear probing technique, which will be discussed later.

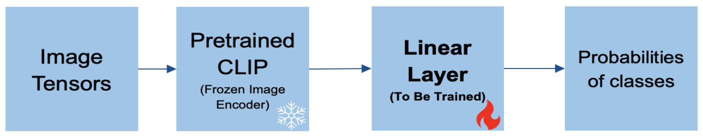


Figure 7: Linear Probing Technique Procedure[14]

Linear Probing

Our experimental approach involved utilizing the linear probing technique to leverage the performance of the pre-trained CLIP model. To adapt the model for our classification requirements, we appended additional layers at the end of the original model. The dimension of these additional layers depends on the image encoder used and the number of classes. For example, with the Vision Transformer (ViT) encoder, the input dimension is 768, and the output size corresponds to the number of classes, such as 2 for a binary task or 3, 4, and so on for multiple tasks.

Figure 7 illustrates the comprehensive procedure. Initially, the image tensors are processed through the pre-trained CLIP model and then passed through the linear layer. Subsequently, predictions are obtained and parameters of the linear layer are updated, while parameters of the pre-trained CLIP model remain unchanged. This differentiation is visually represented by placing a snowflake symbol on the frozen image encoder and a fire symbol on the active linear layer, symbolizing the concept of linear probing or transfer learning.

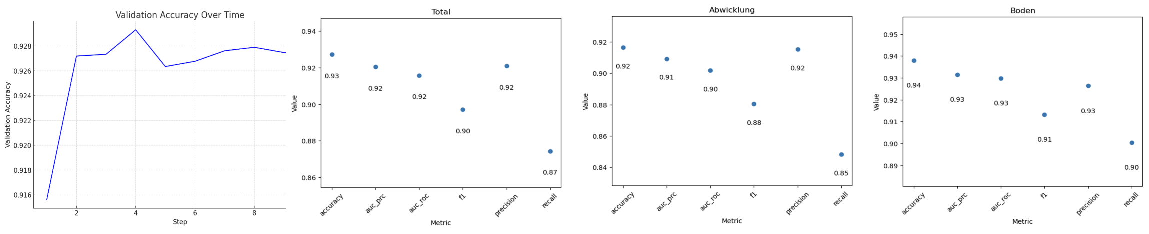
This approach allows us to harness the impressive generalization capabilities of the large CLIP model and adapt it to our specific classification needs using simple linear layers. The linear probing technique proves to be easy to train and has yielded impressive results in our experiments.

Pipeline Structure

The CLIP pipeline mirrors the structure of the baseline pipeline, with two primary distinctions. First, in the data preparation component of the CLIP pipeline, we employ a CLIP-specific preprocessor to preprocess the data. This preprocessor is loaded simultaneously with the pre-trained CLIP model, which offers two types of image encoder to choose, Vision Transformer (ViT) and Residual Convolutional Neural Network (ResNet50). Second, there was no deployment component in CLIP pipeline, because we decided to keep it as a potential candidate for future endeavors. Detailed explanations for this decision will be provided in the subsequent section.

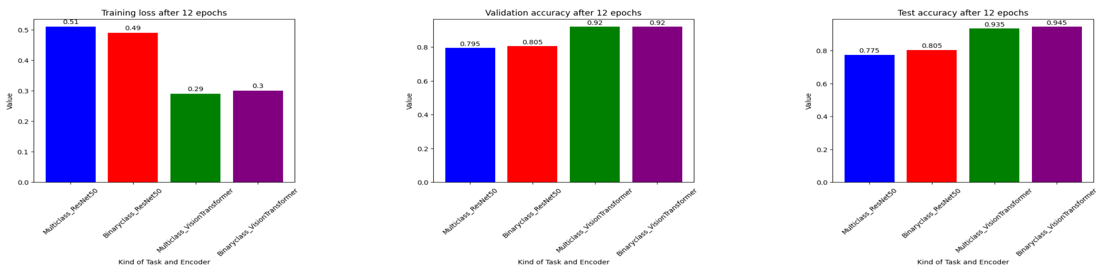
3.2.2 Results

While the CLIP pipeline has yielded surprising results as shown in Figure 8, it has also presented certain challenges, which will be presented subsequently.



(a) Accuracy (b) Total (c) Side View (d) Top View

Figure 8: Metrics of CLIP pipeline



(a) Training Loss (b) Validation Accuracy (c) Test Accuracy

Figure 9: Comparison between ViT and ResNet

Firstly, we observed that ViT as an image encoder consistently outperformed ResNet50 in various classification tasks. This highlights the superior performance of ViT in our specific classification task as demonstrated in Figure 9.

Moreover, the pre-trained CLIP model has demonstrated robust representation learning capabilities on our classification task through the feasible and efficient linear probing technique. However, limitations in performance of CLIP arose from the simplified labeling mechanism, which employed only numerical values (0, 1, 2, etc.). Notably, the task of classifying detergent images proves to be relatively simple, allowing baseline model and CLIP to approach peak performance without showing significant differences. To achieve further improvements, hyperparameter fine-tuning or training on larger datasets can be considered.

It should also be noted that fine-tuning the pre-trained CLIP model requires substantial computational resources. For instance, training or running the whole pipeline on an NVIDIA Tesla T4 GPU took nine hours in our past experiments. As such, employing a more efficient transfer learning technique, like adapter for transformer blocks, could prove beneficial. Within the scope of this project, the CLIP model and the baseline model demonstrated similar performance levels on our classification task. Therefore, due to its lower consumption of computational resources, we opted to deploy the baseline pipeline. Despite this, considering CLIP as an alternative provides us with the opportunity to tap into its potent representation learning capabilities. This potential customization allows us to address more complex scenarios or some multi-modal tasks in the future, thereby maximizing the utility of CLIP for our future projects.

3.3 Multi-class Classification Pipeline

The re-trainable multi-class pipeline is developed in Azure ML and aims to solve the problem of multi-class classification for 3 classes. The pipeline serves as a baseline and consists of two major components. In the future, additional functionality might be incorporated into the pipeline. For instance, deployment component can be seamlessly integrated if deemed necessary.

3.3.1 Modeling

Data preparation

The data is split into three classes: *clear*, *phase*, and *other*. The *phase* class combines all images with different types of 2-phase and 3-phase images. The *other* class is a class of all images that belong to neither clear nor phase classes.

Notably, for this task only *side views* were used for training since they contain the majority of information that can be captured by a model. However, the idea of using both images simultaneously in the training phase might be promising and needs to be researched more thoroughly.

As already mentioned in Section 2.1, one of the main problems faced in data preprocessing phase is a serious class imbalance: class *clear* has 1606 instances, class *other* has 2699 instances, and class *phase* only 125 instances. Therefore, *phase* class is heavily imbalanced in comparison with two other classes and this problem should be addressed in the training phase.

Model

The model is a CNN, suitable for the task complexity. The image is passed 4 times through a sequence of layers: Convolutional $\longrightarrow$ Batch Normalization $\longrightarrow$ ReLU $\longrightarrow$ Max Pooling. Then the resulting tensor is transformed into a vector and passed through two linear layers with a Dropout layer between them. The second linear layer maps the vector to the vector with the logits corresponding to the classes.

Pipeline Structure

- **Train Component**

  The training component is similar to the same component in binary classification pipeline, but there are some significant differences with regard to imbalanced dataset problem:

  - When loading data, a batch sampler is used. It oversamples instances from the underrepresented class by assigning weights to the classes.
  - During training, the Cross Entropy Loss with weights as a loss function selected. Weights are assigned to reflect the number of samples in each class. Therefore, model gets fined more significantly if it fails to correctly classify samples from the imbalanced class.

- **Test Component**

  The test component is identical to the scoring component in the binary classification pipeline. The only difference is macro averaging of precision, recall, and f1-score. Macro averaging is used to take all classes into account equally, which is especially important when dealing with imbalanced dataset. To facilitate the model's performance monitoring, MLFlow is used in both components to log metrics, model parameters, and the model itself.

3.3.2 Results

Validation Results

The model was trained for 15 epochs, and its performance was evaluated after each epoch on the validation dataset. Figure 10 displays the validation loss and metrics after each epoch.

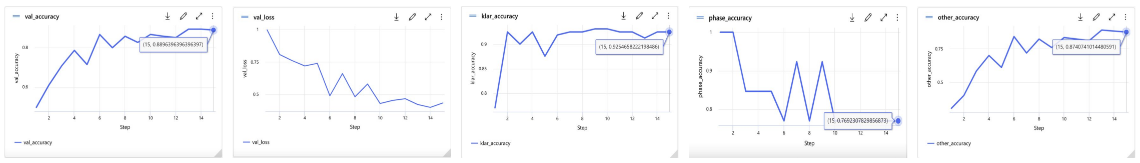


Figure 10: Validation Results

Test Results

The model's performance is evaluated on unseen data that constitute 20% of all data. Below is the Table 11 with aggregated metrics and Figure 12 displays the confusion matrix.

Precision (macro avg)	0.67
Recall (macro avg)	0.75
F1-Score (macro avg)	0.69

Figure 11: Test Results

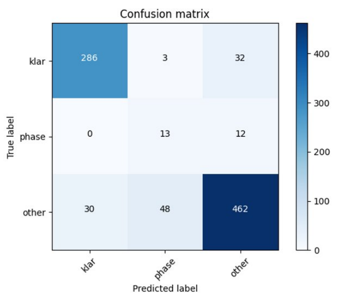


Figure 12: Confusion Matrix

3.4 Image Segmentation

As can be seen in the images of the project dataset, detergent formulations might generate several regions with different characteristics. These varying segments, if further analyzed, can help to enable a more intuitive understanding of the detergent samples' specific attributes and variations. Therefore, various image segmentation methods have been investigated in this project to segment the images into different regions on the vertical axis and to finally investigate the individual regions further in terms of their observable characteristics. Representing each image with its quantifiable image descriptors is useful to remap the detergent samples to the current labels and to eventually identify new and more accurate human interpretable labels.

Detergent properties such as cloudiness or bubbles can be manually detected by human experts. However, the primary objective of combining image segmentation and featurization is to automate the process of quantifiable feature extraction, enabling a more efficient and objective approach to analyze detergent properties. By leveraging these techniques together, the motivation is to expand the range of perceptible features and extract meaningful information from images, enabling more comprehensive and insightful analysis of the data.

3.4.1 Segment Anything Model

Segment Anything Model (SAM)[4], developed by Meta AI, is an innovative promptable segmentation system that exhibits zero-shot generalization capabilities to unfamiliar objects and images. SAM produces high quality object masks from input prompts such as points, texts or boxes, and it can be used to generate masks for all objects in an image. It has been trained on a dataset of 11 million images and 1.1 billion masks, and has strong zero-shot performance on a variety of segmentation tasks.

The presented visual representation in Figure 13 showcases the fundamental framework, wherein the scissor image serves as the input for the ViT-H image encoder. This encoder, equipped with 632 million parameters, processes the image once, generating an image embedding. Concurrently, an auxiliary prompt encoder with 4 million parameters assimilates input prompts, encompassing points, boxes, or textual elements. Finally, a streamlined transformer-based mask decoder with 4 million parameters employs the image embedding and prompt embeddings to predict object masks. The resulting output comprises three masks: one encompassing the entire scissor, another encompassing the complete handle, and the last one exclusively representing the right handle.

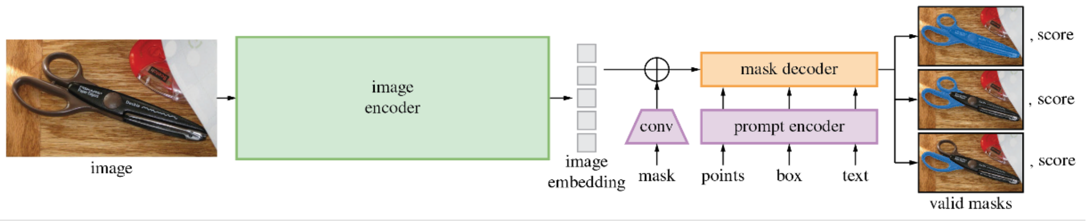
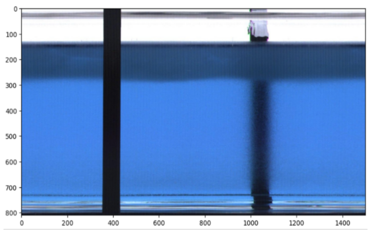
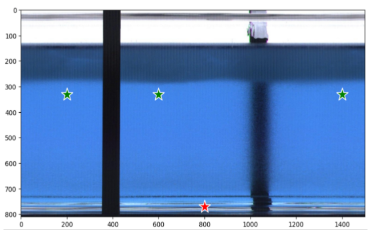


Figure 13: a new AI model SAM from Meta AI that can cut out any object, in any image, with a single click [4]

Prompting Mask Generation

In our experiment, we evaluated the model's performance on our dataset using two different methods for mask generation. Specifically, we employed segmentation through points prompt and the automatic mask generator, as mentioned earlier. In Figure 14, the process of selecting the middle part is demonstrated. Multiple points are chosen on this section, which are then inputted to the model. Each point is accompanied by a label of 1 denotes a foreground point visually represented as green star, while label of 0 represents a background point as red star. It is also possible to input a single point. This work can be efficiently completed with just a few lines of code, encompassing essential steps such as loading the model and predictor, generating image embeddings, selecting object points, and generating masks. The Figure 14 and Figure 15 exemplify the effective segmentation achieved, where the positive area is accurately delineated by a red mask. Notably, the segmentation exhibits remarkable precision, as the boundary lines are not merely straight, but gracefully curved, enhancing the visual fidelity of the segmentation.


(a) Before segmentation


(b) After prompting

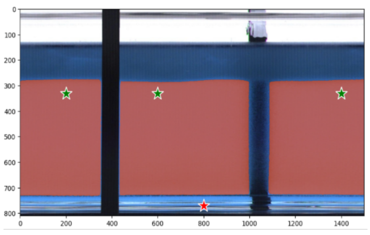
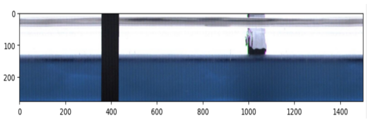
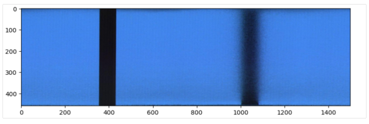

(c) After segmentation

Figure 14: Single object mask generation through prompts


(a) Part 1


(b) Part 2

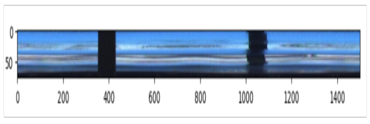

(c) Part 3

Figure 15: Multi-Phase Detergent Image Segmentation utilizing SAM

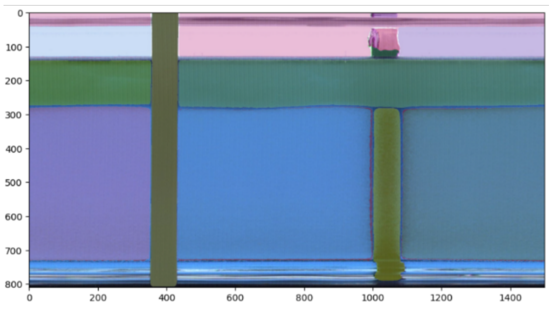
Automatic Mask Generation

In conjunction with points prompts, the SAM framework extends its capabilities to include automatic mask generation. Leveraging the efficient prompt processing of SAM, an extensive sampling of prompts can be conducted across an image to generate masks for the entire image.

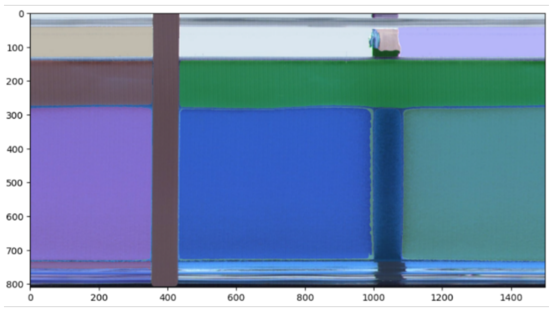
The class `SamAutomaticMaskGenerator` implements this capability. It works by sampling single-point input prompts in a grid over the image, from each of which SAM can predict multiple masks. Then, masks are filtered for quality and deduplicated using non-maximal suppression. To utilize this functionality, one can simply specify the path to the SAM checkpoint. By invoking the `generate` function on an image, masks can be generated automatically.

The automatic mask generation process includes several adjustable parameters that govern the density of point sampling and determine thresholds for removing low-quality or duplicate masks. Additional options allow for further improvement of mask quality and quantity, such as running prediction on multiple crops of the image or postprocessing masks to remove small disconnected regions and holes.

The resulting segmentation outcomes are depicted in Figure 16, wherein Figure 16a showcases auto-segmentation without additional configuration, while Figure 16b demonstrates auto-segmentation with several fine-tuned hyperparameters.



(a) Auto-segmentation



(b) Auto-segmentation with parameters

Figure 16: Automatic Mask Generation

Results

Undoubtedly, SAM exhibits exceptional performance when utilizing points prompts, showcasing its remarkable flexibility. However, when applied to our detergent dataset, SAM's automatic mask generation feature falls short of expectations. To be precise, it demonstrates overly precise segmentation, often dividing the image into excessively small regions, even in cases where the differences between these regions are minute. Consequently, its feasibility for our specific use case is diminished.

In conclusion, SAM has demonstrated an incredible ability to perform segmentation across a wide range of tasks. Nevertheless, due to the distinctive characteristics of our detergent dataset and the high computational cost involved, which amounts to 0.15 seconds for image encoding on an NVIDIA A100 GPU and even longer duration on Azure ML GPU clusters, we have made the decision to proceed with Edge Detection.

3.4.2 CAM/Grad-CAM

Class Activation Mappings (CAMs)[15] is a method that, given a CNN model, allows understanding which regions in an image are important to make a classification decision. This is done by applying a Global Average Pooling Layer to the outputs of the final Convolutional Layer in the network, followed by a weighted sum where the weights are the same as the weights in the final Fully Connected Layer. This allows us to create a heatmap where the intensity at each location is the weighted sum of the activations at that location. This will reveal where in the image the model is looking at to make a decision about a particular class.

The aim is to check the heatmap for our Multiclass model for the multi-phase images. Ideally, this will lead to a heatmap where the different phases are separated very clearly, conducive for segmenting the original image into regions corresponding to the different phases. However, this method is very limited in its applications since it requires a very specific type of model architecture. This is why we instead used a modification of this technique, called Grad-CAMs.

Gradient-weighted Class Activation Mappings (Grad-CAMs)[11] is a method that is an extension of CAMs, where it is applicable to models with different architectures, unlike CAMs. It no longer requires a model to have a Fully Connected Layer in order to generate a heatmap. Grad-CAM works by calculating the gradient of the class score in the final Convolutional Layer, with respect to the feature maps. These gradients are averaged to yield a weight, which is interpreted as the *importance* of that particular feature map for the class prediction. Taking a weighted sum of the feature maps using these weights and applying a ReLU function to remove features that have a negative effect on the specific class, we can generate a heatmap. This heatmap will visualize the important regions in the image (as per our model) for a given class.

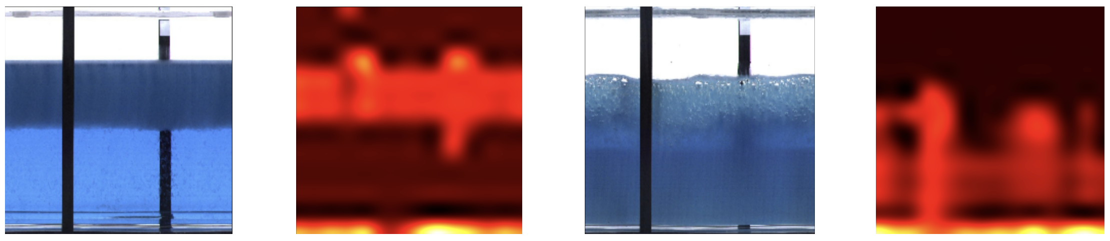

(a) Original Image (b) After Grad-CAM (c) Original Image (d) After Grad-CAM

Figure 17: Comparison of Original Images and corresponding Grad-CAMs

Results

The results after using the model in the Multiple Classification Pipeline and applying Grad-CAM with respect to the multiphase image classes can be seen in Figure 17. It is clear that while the model accurately assigns importance to the different phase regions, the distinct phases are not the only parts of the image the model considers when making a decision. We can see in particular that the black bars play a big role in the decision process of the model as well. This makes it difficult to automatically decide how to segment a given image.

In conclusion, even though using Grad-CAMs is a unique and promising approach for segmenting images, it is not clear how to accurately and consistently get good results. For our particular task, there are simpler ways such as Edge Detection and OpenCV methods that yield better results than using Grad-CAMs.

3.4.3 OpenCV

This method uses OpenCV library[2], which provides a range of functions and algorithms that facilitate image processing and segmentation operations.

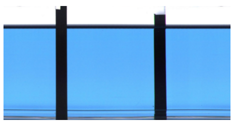
Starting by grayscaling and blurring, **adaptive thresholding** is applied in the subsequent step to convert a grayscale image into a binary image. It adjusts the threshold value for each pixel based on the local neighborhood's intensity distribution. This adaptive approach allows for effective thresholding even in the presence of varying illumination and contrast conditions within the image.

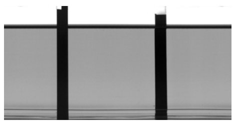
After the thresholding step, the **Canny edge detection** algorithm is employed to identify edges within the binary image. Image segmentation aims to divide an image into meaningful regions or objects. The boundaries of these regions are often characterized by edges, where there are significant intensity changes or transitions between different regions. Canny edge detection helps identify these edges, which can then be used as cues for segmenting the image.

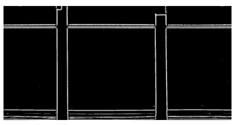
Next, the utilization of a morphological operation, specifically **dilation**, serves to emphasize the detected edges from the previous step by thickening the lines. This process enhances the robustness of edge detection and contributes to better-defined structures in the image.

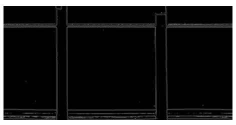
The final step is **contour extraction**. Contours represent continuous curves that outline the boundaries of objects within the image. It allows the detection of bounding boxes for different regions within the image. Finally, image is segmented on the vertical axis by further processing the detected bounding boxes, such as filtering out very small regions or filtering out the y-positions that are too close to the image borders.

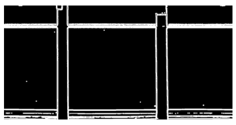
Figure 18 illustrates the operations applied to a *clear* image. The resulting two segments, along with their respective ratios of occupied area to the total liquid area, are displayed in Figure 18g.

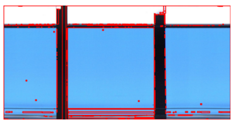

(a) Clear Image

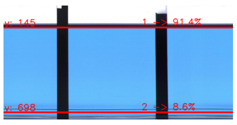

(b) Grayscale, Blur


(c) Threshold


(d) Canny


(e) Dilation


(f) Regions



(g) Final result

Figure 18: OpenCV segmentation operations applied on a clear image

[2] `https://docs.opencv.org/4.8.0/d7/dbd/group__imgproc.html`, [accessed 12.07.23]

This method employs classical image processing algorithms in a simple and efficient manner. It is able to detect regions smaller than a detergent segment, such as bounding boxes around the small bubbles or sediments. These regions provide valuable information for extracting quantifiable features from the segments. In the image segmentation part, these smaller regions are filtered out, and the focus is solely on detecting horizontal edges in the image. Therefore, choosing a more straightforward horizontal edge detection approach is deemed more beneficial, allowing to concentrate on this specific task.

The next section presents the final decision on the image segmentation method.

3.4.4 Edge Detection

After a closer examination of the images and what we exactly want to segment, we observed that the segmentation task mainly involved identifying and separating regions based on horizontal edges. Therefore, we decided to leverage traditional computer vision and image processing methods to develop an algorithm specifically tailored for this purpose. The segmentation algorithm consists of the following steps:

Horizontal edge detection

We applied the Sobel operator to detect horizontal edges in the images. This operator calculates the gradient magnitude of the image to highlight regions with significant horizontal variations.

Thresholding

Using a suitable threshold value, we converted the gradient magnitude image into a binary image. This step allowed us to emphasize the regions containing horizontal edges while suppressing other image details.

Filtering

Due to the nature of the detergents and having variety of cases, there were horizontal edges detected on the whole y-axis, so we decided to only consider y-cutoffs where there were a minimum number of pixels considered as edges on the horizontal axis. Thus refining the binary image and enhance the quality of the detected edges.

Grouping

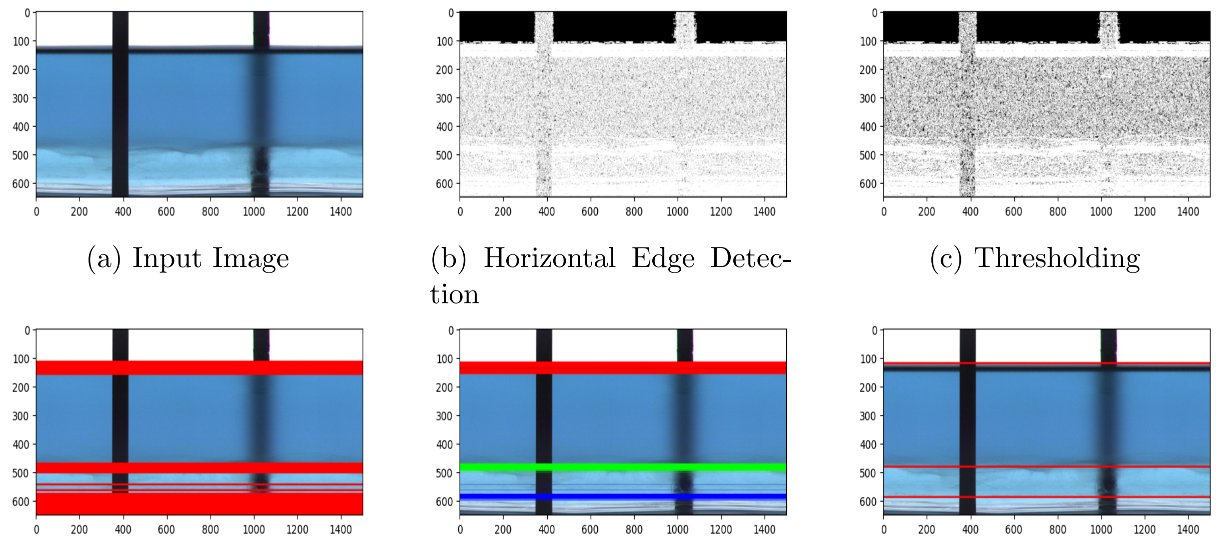
In this step, we wanted to combine the horizontal edges, which are very close to each other, into a single edge. To do so, we incorporated a marching method to group the edges together, in which we used two different criteria. If one of them is satisfied, then they would be added to the same bin:

1. The vertical distance between the two edges is lower than a threshold.
2. The vertical distance between the current edge and the smallest index in the bin is lower than another threshold.

Aggregation

Finally after grouping the edges together, we wanted to combine them into one edge. After experimenting with different methods, we came to the conclusion that the mode is best suited for our case across different classes, which is only considering the y-cutoff that has the most number of pixels considered as a horizontal edge for each bin.

The developed segmentation algorithm enabled us to partition the images into meaningful segments based on horizontal edges and segment characteristics, providing additional insights and information for further analysis and interpretation. An illustrative representation of our segmentation process can be observed in Figure 19, providing a visual insight into each step of the method.



(a) Input Image (b) Horizontal Edge Detection (c) Thresholding

(d) Filtering (e) Grouping (f) Aggregation

Figure 19: Segmentation steps

3.5 Featurization

After segmenting an image, the goal is to extract meaningful information from each of the image segments, in order to utilize them in future tasks such as remapping labels, or in time series predictions. Crucially, the extracted information, or features, should be human-interpretable. This can provide valuable insight to scientists by allowing them to check how the features of an image are being used to make predictions, or classify images. Extracting numerous distinct features from images will facilitate creating a comprehensive feature vector that encompasses enough information about the images, to be used in later tasks. To achieve featurization, the image segments are fed into a `Characterizer` block, which has `Featurizers` inside that extract the relevant features. The `Characterizer` then outputs extracted features and their descriptions for each image segment.

3.5.1 Featurizers

A Featurizer aims to extract a specific type of feature (e.g. related to color or opacity) from an image segment. In order to achieve this, each Featurizer is implemented in a specialized way that allows it to extract the required feature in a human-interpretable manner.

Phase Width Featurizer

The phase width featurizer returns the width for a given image segment. Since each image is split into phases after segmentation and then each segment is passed to the characterizer-featurizer pipeline, the final feature vector will contain information about the width of each phase for each image. This feature is very useful, since it allows to distinguish classes with different number of phases right away. Combining this feature with color-related features, it is very likely possible to differentiate between classes with the same number of phases (for example, sedimentation and 2-phase). Last but not least, this feature is important for time series prediction, as the phases' width change over time.

Opacity Featurizer

The purpose of using this featurizer is to determine the opacity level of the image segment by analyzing the visibility of the black bars in the detergent tank.

The presence of one black bar in the foreground, unaffected by the liquid behind it, is expected to result in a high ratio of black pixels. The second black bar, located in the background and affected by the liquid in front of it, will have its visibility influenced by the liquid's opacity. If the liquid is highly opaque, the black bar may be obscured. If the liquid is translucent, the black bar will appear blurry with reduced number of black pixels. A clear liquid will allow the black bar to remain clearly visible with a high ratio of black pixels. The visibility of the black bars thus serves as an indicator of the liquid's opacity level.

It is important to note that this featurizer is specific to the provided image dataset, assuming a consistent experimental setup during data acquisition. The assumptions include images taken from the same distance, the use of the same tank, at least two black bars in the image, and the black bars having the same width and distance between them.

The process involves thresholding a grayscale, blurred image segment and then finding contours to detect the bounding boxes. Further processing is performed to extract the black bar locations using the bounding boxes and the dataset specific parameters. The featurizer returns two values: the number of detected black bars and the ratio of pixels with intensities below a specified threshold value to the total number of pixels within the black bar area.

Figure 20 describes the application of the opacity featurizer on a cloudy image. As can be seen in the first image, a detergent image labeled as *cloudy* is not completely clear and one can observe the blurriness on the right black bar. The middle image in Figure 20 displays the detected areas highlighted in red. It can be seen that the first black bar is successfully detected, while the detection of the second bar is incomplete. The third image shows the location of the first black bar, determined through the contour detection process, alongside the position of the second bar, determined based on the dataset-specific parameters. Notably, pixels falling below a designated threshold value are distinctly highlighted in orange color.

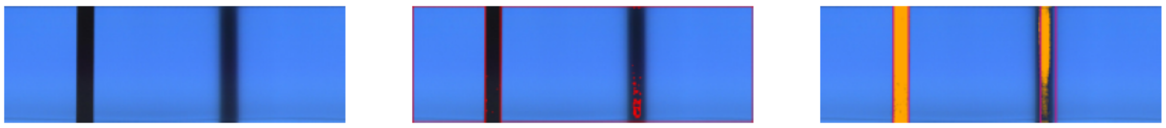


Figure 20: (left) A cloudy image, (middle) Detected regions, (right) Orange color showing pixels below a threshold value (here 40), pink lines showing black bar locations

Color Featurizer

The Color Featurizer aims to extract color-related features from a given image segment. It looks at the distributions across all color channels (RGB) of the image, and outputs a vector containing the means and standard deviations of these color distributions.

Importantly, the images in our dataset have black bars in front of and behind the liquid. We should not take these black pixels into account when calculating statistics from the color distributions across channels, since they are not part of the true color distribution of the liquid. To deal with this issue, the Color Featurizer first creates a binary mask for the dark pixels and applies it to each color channel. Effectively, this ensures the featurizer only works with the actual liquid and not the black bars.

More statistics from the color distributions can be considered in the Color Featurizer as features, such as the skewness or the entropy of the distributions.

Other Possible Featurizers

In addition to the aforementioned featurizers, we have identified several potential featurizers that may be of relevance for future requirements, as listed in table 2. Any new feature can be added easily by creating a new Featurizer class and passing it to the Characterizer.

Additional Featurizer	**Description**
Color Histogram	Analyzing the distribution of colors in the image using color histograms can provide information about the dominant colors present in the detergent image.
Texture Analysis	Extracting texture features can help capture the surface properties of the detergent, such as smoothness or roughness. Texture features can be obtained using methods like Local Binary Patterns (LBP) or Gabor filters.
Shape Descriptors	Detergent images often have specific shapes or contours that can be quantified using shape descriptors like moments, Fourier descriptors, or scale-invariant feature transform (SIFT).
Size and Area	Measuring the size and area of detergent objects within the image can provide quantitative information about their dimensions.
Edge Detection	Identifying and quantifying edges within the detergent image can reveal important structural information, such as the boundaries of the detergent or the presence of patterns.
Object Count	Counting the number of detergent objects present in the image can be a useful quantitative metric for assessing density or concentration.

Table 2: Potential features of detergents

3.5.2 Characterizer

The `Characterizer` class is instantiated by passing a list of `Featurizer` objects to it. This list of featurizers corresponds to the desired features to be extracted from the image segments. It then takes in the segments of an image as a list, to output the extracted features from each image segment along with their descriptions as a dictionary. This can later be transformed into a single feature vector for a given image by simply concatenating the feature vectors of the image segments.

Since in general images are segmented into a different number of regions, it is important to have a consistent feature vector size for all images. This is done in the `Characterizer` by padding the the feature vectors with zeros, for the images with fewer segments than the maximum number of segments.

3.6 Feature Remapping

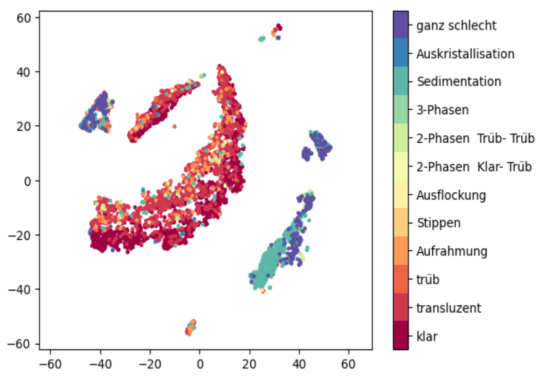
In this section, we explore the process of identifying mislabeled data points after successful image segmentation and feature extraction, as discussed in the previous sections. To achieve this task, we investigated three distinct techniques: K-Means clustering, t-Distributed Stochastic Neighbor Embedding (t-SNE)[5], and Uniform Manifold Approximation and Projection (UMAP)[6].

3.6.1 K-Means Clustering

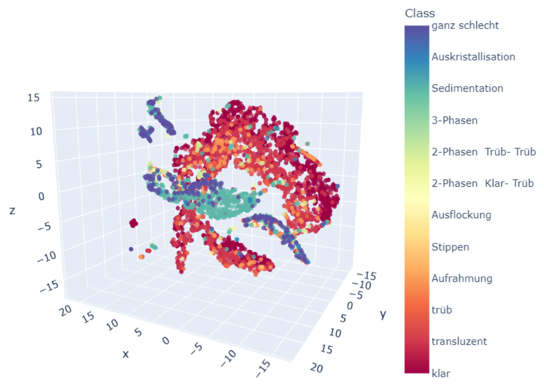
K-Means clustering is a classic unsupervised learning method that partitions the dataset into K clusters based on the similarity of data points to the cluster centroids. By iteratively optimizing cluster assignments, K-Means aims to minimize the within-cluster variance, thereby enabling us to identify potential mislabeled data points that do not have same labels as neighbors. However, in our experiments, we observed that K-Means clustering did not consistently show a cluster distribution similar to our original labels. This suggests that K-Means may not be the most suitable technique for identifying mislabeled data points in our dataset.

3.6.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

T-SNE is a nonlinear dimensionality reduction technique that excels at visualizing high-dimensional data in a lower-dimensional space. In our experiments, for the 2D t-SNE experiment, we directly used all features to project the data into a 2D space. In contrast, for the 3D t-SNE experiment, we initially employed Principal Component Analysis (PCA) to reduce the dimensions before applying t-SNE. This allowed us to fine-tune the t-SNE parameters and gain valuable insights into potential mislabeling issues. By projecting the data into reduced dimensions, we visually inspected potential mislabeled points that deviate from the expected clusters. The results of both can be seen in Figure 21. As can be seen in the figure, techniques were able to produce better clustering between the different classes and also as can be seen some mislabelled samples were actually detected.



(a) 2D TSNE on the Detergents Dataset



(b) 3D TSNE on the Detergents Dataset

Figure 21: Applying TSNE on the Detergents Dataset

3.7 Uniform Manifold Approximation and Projection

The final method we experimented with is UMAP which is another dimensionality reduction technique that excels at preserving global and local data structures. By constructing a topological representation of the data, UMAP facilitates the detection of mislabeled points through their relative positions in the lower-dimensional space. Similar to t-SNE, UMAP also allows for visualizing the data in a reduced dimensionality, providing additional perspectives to identify potential mislabeling issues as can be seen in Figure 22.

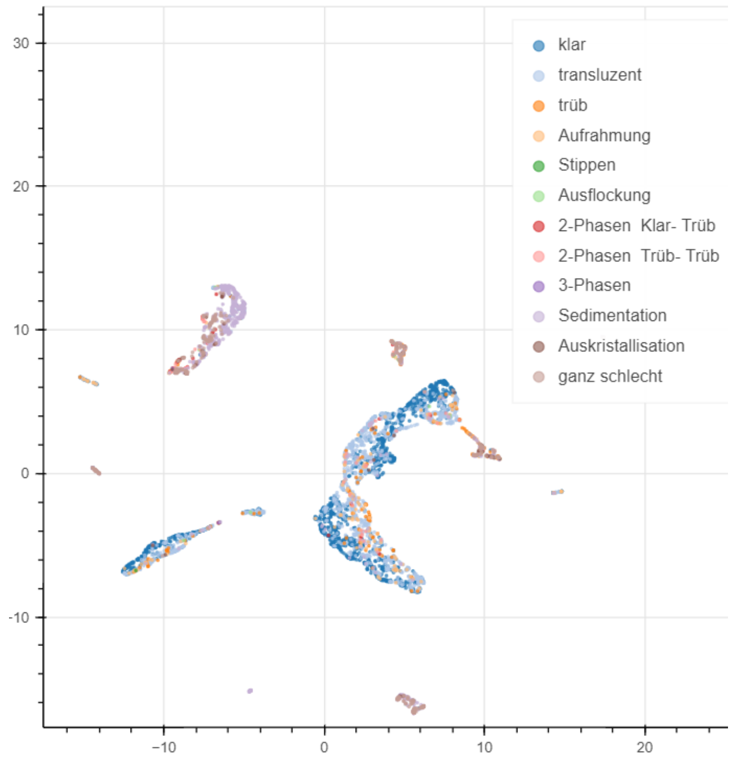


Figure 22: Applied UMAP on the Detergents Dataset

The algorithm was able to produce the best clusters in addition to also detecting more outliers, and when comparing with the original labels, some of the outliers were actually mislabeled, however we are certain that there is still much more room for improving the clusters, since after all we cannot distinguish between the 12 different classes, but by increasing the quality of the extracted features and even adding more *featurizers*, we believe that this is defiantly a promising path.

4 Discussions

Our work in this project provides numerous improvements to the existing methods used by the experts. Firstly, the pipelines all proved to be effective on classifying the images correctly. They can be used to automate the labeling of the detergents, freeing the scientists from having to do so. Secondly, the mislabeled images present problems to any model used on the dataset. Our work on extracting features and remapping the labels can be beneficial to correct the labels of the mislabeled images so that this process yields a higher-quality dataset.

The solution we developed offers the flexibility to be easily extended and applied to different products beyond detergents. The complete end-to-end system, encompassing image classification, segmentation, feature extraction, and feature remapping, can be adapted to various industries and domains. By leveraging the underlying principles and methodologies of our pipeline, it becomes feasible to train and retrain the system with new datasets specific to different products. This adaptability enables Henkel experts to utilize the same reliable and efficient solution for automated image labeling across multiple product categories, enhancing productivity and decision-making in various quality assessment processes.

4.1 Future Works

Building upon our accomplishments in image recognition, classification, segmentation, and feature extraction, the next significant objective is to leverage the extracted feature vectors for performing time series prediction[12]. This will facilitate forecasting the changes in liquid properties over time, empowering Henkel experts to assess detergent stability, expedite research progress, and enhance control over liquid production quality. To accomplish this, we will integrate the image feature vectors with existing time series data and employ various machine learning or deep learning techniques, such as recurrent neural networks (RNNs)[10], long short-term memory (LSTM) networks[2], or transformers[13]. By combining image feature vectors with time series data, we can effectively capture both visual and temporal information, potentially elevating prediction accuracy.

Following our comprehensive discussion on the deployment of the CLIP pipeline, we intend to enhance our image dataset by incorporating meaningful textual labels for each image. In this way, we plan to activate the text encoder in the CLIP model to evaluate its impact on performance improvement. Furthermore, we recognize the importance of hyperparameter fine-tuning, which necessitates a substantial amount of computational resources. Therefore, we are prepared to allocate adequate resources to this endeavor in order to optimize our model's performance.

Furthermore, we can consider exploring the use of Factorized Variational Autoencoders (VAEs)[3] to extract meaningful and interpretable features from the images. This approach has the potential to enhance the interpretability of our system by capturing essential visual attributes and disentangling the factors of variation present in the images. By integrating factorized VAEs into our solution, we can improve the quality assessment and characterization process by leveraging these extracted features. It would enable us to gain a deeper understanding of the visual characteristics that drive product quality, facilitating more informed decision-making in terms of developing top-notch *Featurizers*, moreover, providing better remapping.

5 Conclusions

In conclusion, this project report highlights the implementation of different approaches to achieve our project goal. Our primary objective was to provide Henkel experts with a customized machine learning pipeline that is readily available and can be effortlessly used. By developing an automated image labeling system, we aimed to optimize Henkel's detergent quality assessment process, reduce manual workload, and improve decision-making precision. Our focus was on creating a versatile classification pipeline capable of accurately labeling detergents in both binary and multiclass scenarios. Leveraging image analysis and machine learning techniques, we aimed to build a robust and efficient system that objectively assesses detergent quality and categorizes them into different classes. This includes distinguishing between *good* and *bad* detergents (binary classification) as well as classifying them into various categories (multiclass classification).

Moreover, we utilized CLIP, a state-of-the-art image-text model, for binary and multiple classification. This allowed us to leverage the power of natural language processing, image understanding and the zero-shot learning to accurately categorize the input data. The integration of CLIP into our pipeline enhanced the classification accuracy and expanded the capabilities of our system, but we decided to keep it as an optional pipeline for more complex datasets.

Additionally, we wanted to provide more insights on how the classification decision was made, particularly for the lab experts, therefore we recognized the importance of interpretability in our solution, and as a first step we conducted an investigation into multiple segmentation techniques to identify the most effective approach for segmenting the images into different phases. This analysis allowed us to select the most suitable technique that provided accurate and meaningful segmentation results, serving as valuable inputs for subsequent stages of our work.

Furthermore, we developed the *Characterizer* and multiple *featurizers* to extract relevant features from the segmented images. These features served as valuable inputs for capturing important characteristics and patterns within the data.

Finally using the features extracted, we investigated different techniques to be able to classify mislabeled data points and outliers and were able to detect some mislabeled data points. However we still believe that there is more room for improvements with regards to that part.

References

[1] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[2] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[3] Hyunjik Kim and Andriy Mnih. *Disentangling by Factorising.* 2019. arXiv: `1802.05983 [stat.ML]`.

[4] Alexander Kirillov et al. “Segment Anything”. In: *arXiv:2304.02643* (2023).

[5] Laurens van der Maaten and Geoffrey Hinton. “Viualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605.

[6] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* 2020. arXiv: `1802.03426 [stat.ML]`.

[7] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks.* 2015. arXiv: `1511.08458 [cs.NE]`.

[8] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[9] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision.* 2021. arXiv: `2103.00020 [cs.CV]`.

[10] M. Schuster and K.K. Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. DOI: `10.1109/78.650093`.

[11] Ramprasaath R Selvaraju et al. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 618–626.

[12] Ahmed Tealab. “Time series forecasting using artificial neural networks methodologies: A systematic review”. In: *Future Computing and Informatics Journal* 3.2 (2018), pp. 334–340.

[13] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[14] Wikipedia contributors. *Linear probing — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=Linear_probing&oldid=1151911536`. [Online; accessed 12-July-2023]. 2023.

[15] Bolei Zhou et al. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 2921–2929.

Appendix