Pollution Image Classification Using CNNs: Training and Analysis

University examination report for the subject "Artificial Vision"

Lanzarotti Raffaella
Assistant Professor and Researcher
Department of Computer Science of the University of Milan
Milan, Italy
lanzarotti@di.unimi.it

Francesco Ruggeri
Computer Science Master's degree student
University of Milan
Milan, Italy
francesco.ruggeri2@studenti.unimi.it

***Abstract*—In this study, we leverage the power of artificial vision techniques to develop a waste image classification model. The objective is to automatically identify and classify various types of pollutants from digital images, such as bottles, detergents, glass.**

***Index Terms*—Pollution, CNN, Classification**

I. Dataset

A. WaRP - Waste Recycling Plant Dataset

The dataset used in this project was downloaded from Kaggle, a popular platform for data science competitions and datasets. This dataset consists of a collection of 28 distinct categories of images of recyclable products. The dataset provided is split into two subsets: the training and test set, containing 8823 and 1551 entries respectively. Combined, these subsets create a dataset of 10374 images in total.
In figure 1 it is shown a preview of some images of the dataset.

Figure 1. Dataset preview

B. Image dimension distribution

The first analysis is conducted on the widths and heights of the images. In Figure 2, it is possible to see an histogram illustrating the distribution of these values. The distribution shows a skewed pattern.

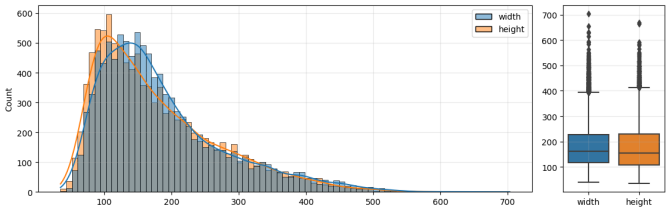


Figure 2. Width and Height distribution

The following statistics have been evaluated on the distribution:

	mean	std	min	25%	50%	75%	max
width	185.75	92.96	40	118	162	228	703
height	178.72	92.71	35	107	154	229	668

Next, the aspect-ratio of the images was examined. The aspect-ratio of an image, which is the ratio of its width to its height, was analyzed to determine a standardized size for resizing during the training phase. Figure 3 provides the histogram distribution, revealing that the majority of images have ratios centered around 1.

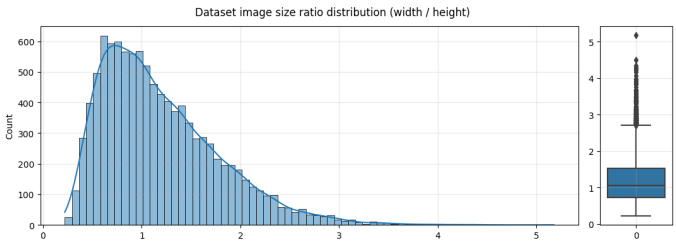


Figure 3. Width/Height distribution

Considering the median value of the width-to-height distribution, which falls within the range of $(162 + 154)/2 = 158$, and taking into account hardware-related considerations, we determined that a resizing dimension of 128 would be a suitable choice for normalizing the image dimensions, both width and height. It is important to note that this chosen dimension of 128 will be applied during the data loading process, where each image will be resized to $128x128$.

C. Class distribution

The figure 4 shows the image label distribution. It can be observed that the dataset is unbalanced, mostly because of the class *bottle-transparent*.

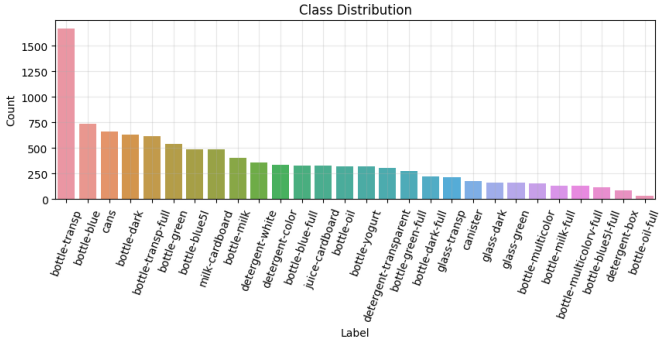


Figure 4. Class distribution

Follows the class distribution among training and test splits, where we can notice that the *bottle-transparent* class is overpopulated in the training set.

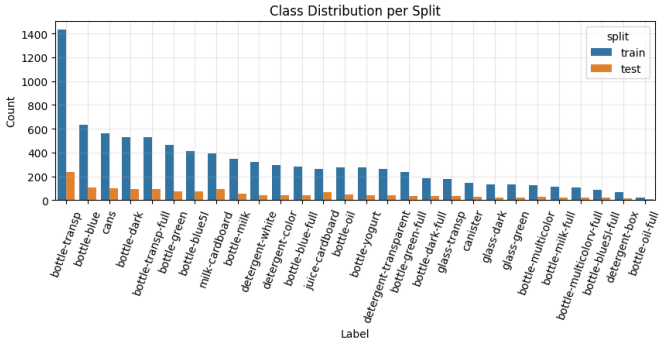


Figure 5. Dataset training/test splits

D. Dataset balancing

Considering classes with high amount of entries, the only outlier class is the *bottle-transp*. In order to balance the dataset, this class will be reduced to the same amount as the second most populated class, which is *bottle-blue*, composed of 634 entries. The most populated class is undersampled exclusively in the training set. The less populated classes are left untouched in order to allow a better learning and generalization of these less frequent categories. The distribution of classes, after applying this data balancing, can be observed in figure 6.

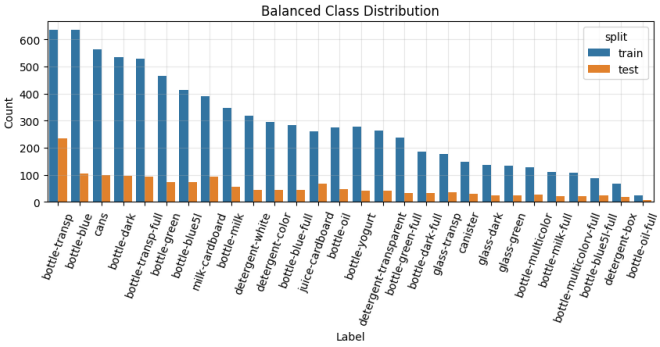


Figure 6. Balanced class distribution

Classes that were downsampled, like *bottle-oil-full*, remained unchanged due to their similarity with the more populated classes.

E. Training, validation and test splits

In order to monitor and evaluate the performances of the trained model during the training process, a validation set has been created. The validation set consists of 10% of the training data (training set only) from each class. Once the splits are created, the new dataset was stored in a different folder, in order to simplify the data loading in future procedures. In figure 7 it is shown the final class distribution.

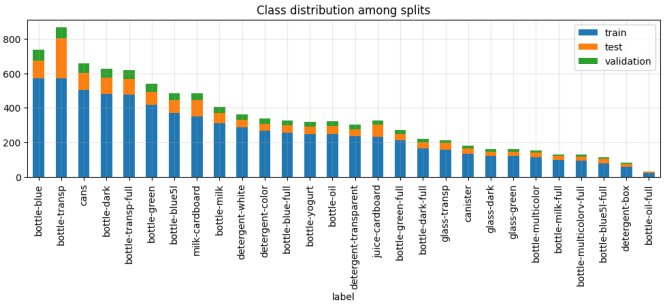


Figure 7. Training, validation, test splits

F. Dataset color normalization

Color normalization is a common operation performed on image datasets because it helps to standardize the color representation across different images. By applying color normalization, variations like lighting conditions or color intensity can be minimized and this allows the model to focus more on relevant features and patterns of the images.

Channel normalization is a type of color normalization that is performed by dividing each value of each color channel by a given set of values. These values are usually derived from popular datasets like ImageNet, but in this academic project they have been evaluated on our specific dataset.

For each image, the mean and standard deviation of each color channel (axis 0, 1, 2) are calculated. The figure 8 and 9 shows respectively the mean and standard deviation distribution of each channel in each image.

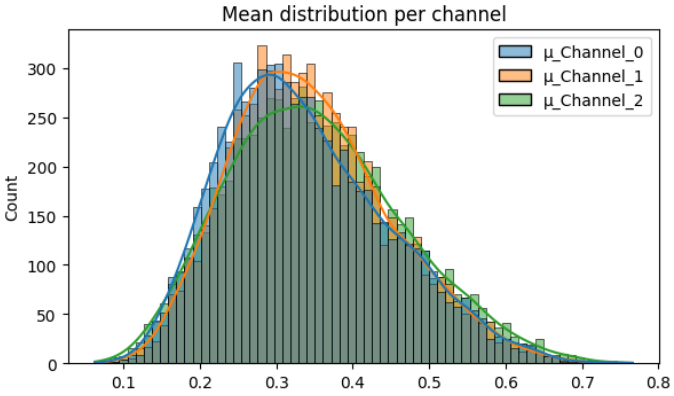


Figure 8. Mean channel distribution

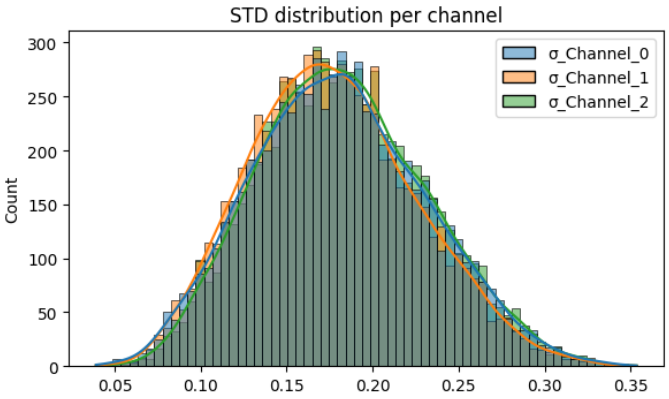


Figure 9. STD channel distribution

After analyzing the calculated means and standard deviations for each color channel across the dataset, it was observed that the majority of the values are encountered around 0.34 and 0.18 respectively. Based on this observation, the final values for image normalization were determined by averaging the means and standard deviations, giving the following final result:

	Channel 1	Channel 2	Channel 3
mean	0.33	0.34	0.35
std	0.18	0.17	0.18

These values obtained for each color channel will be used to normalize the pixel values of the images during dataset loading. This is achieved using the *transforms.Normalize* provided by the *torchvision* library. In the following figure 10 it is possible to observe a sample of normalized images.

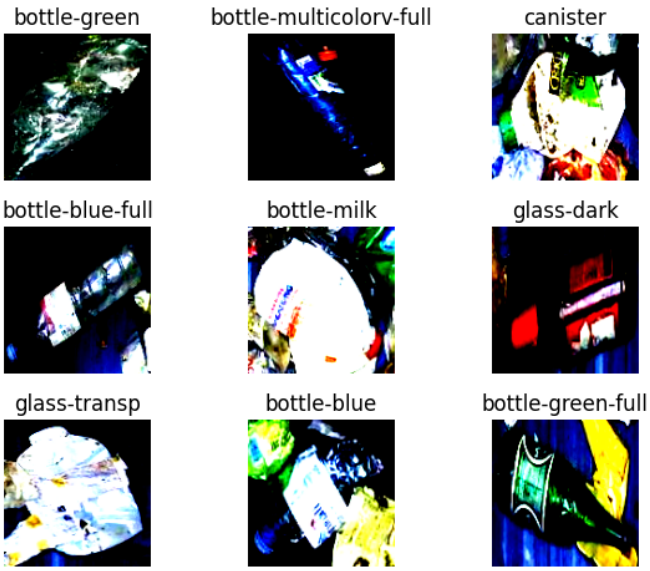


Figure 10. Sample of normalized images

G. Data augmentation

Data augmentation is an effective technique to enhance model training without requiring additional training data. By applying various transformations to the existing images, the model's prediction capacity can be improved because it can generalize better. In this project, seven transformations were analyzed and implemented during the data loading process:

- **Horizontal flip**: flips the image horizontally.
- **Vertical flip**: flips the image vertically.
- **Random rotation**: rotates the image by a random angle between -45 and +45 degrees.
- **Perspective transformation**: changes the image perspective with a distortion scale of 0.3.
- **Color equalization**: equalizes the image colors.
- **Adjust sharpness**: enhances image sharpness using a sharpening filter with a factor of 5.
- **Auto contrast**: applies automatic contrast correction to the image.

In the following figures, it is possible to observe a sample of augmented images.

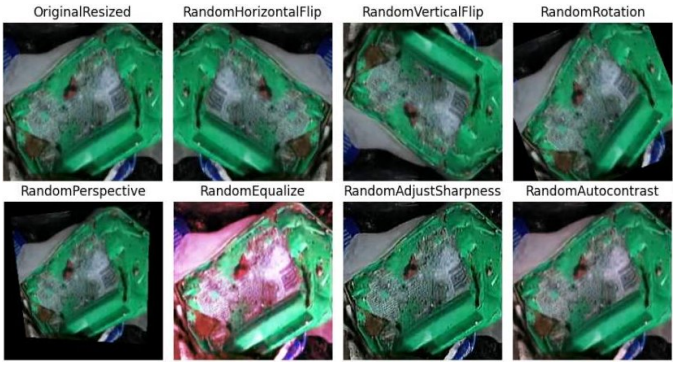


Figure 11. Augmentations applied to a sample image

Figure 12. Augmentations applied to a sample image

Image transformation is applied during the data loader execution to augment the dataset. Each filter has been configured with a probability of 20% to be applied to an image. With a total of 7 filters, the probability of having at least one filter applied is calculated as $1-(1-0.2)^7$, resulting in a probability of approximately 75%. This means that in most cases, at least one filter will be randomly selected and applied to each image during the data loading process.

II. Model

A. CNN Architecture

The model architecture chosen for this project is the Convolutional Neural Network (CNN), which is widely used for image classification tasks in machine learning. CNNs excel in capturing spatial hierarchies and extracting meaningful features from images.

CNNs primarily rely on convolutional operations, where input images are convolved with learnable filters or kernels. This process involves scanning the input image with small windows, performing element-wise multiplication, and aggregating the results. Through this mechanism, CNNs effectively extract features from the input data. By learning and applying these filters during training, CNNs can automatically detect patterns, edges, and textures within the images.

The CNN model for this project is composed of the following layers, which combined are able to learn and extract relevant features from the input images:

- **Convolutional Layer**: this layer applies convolutional filters to the input image, generating feature maps that capture different aspects of the image.
- **Batch Normalization**: batch normalization is employed to normalize the activations of the previous layer, improving the stability and speed of training.
- **ReLU**: ReLU is a commonly used activation function that introduces non-linearity to the model, enabling it to learn complex patterns and make nonlinear predictions.
- **MaxPool2D**: this layer performs down-sampling by selecting the maximum value from each local region, reducing the spatial dimensions and retaining the most important features.
- **Flatten**: the flatten layer reshapes the multi-dimensional output from the previous layer into a vector, preparing it for input to a fully connected layer.
- **Linear Layer**: the linear layer connects the neurons from the previous layer to the neurons in the subsequent layer. It performs a linear transformation on the input data by applying weights and biases to generate an output. This layer helps in mapping the features learned from the previous layers to the desired output, enabling the model to make predictions based on the learned representations.
- **Dropout**: dropout is a regularization technique that randomly sets a fraction of input units to zero during training, preventing overfitting and improving generalization.

Our model architecture consists of two main parts: the feature generation layers and the classification layers.

The feature generation layers are designed to extract relevant features from the input images. They are organized into three similar blocks. Each block includes a convolutional layer followed by the MaxPool operation. This helps capture spatial hierarchies and reduce the spatial dimensions of the feature maps. To improve the stability and convergence of the model, batch normalization is applied after each convolutional layer, and the ReLU activation function is used to introduce non-linearity.

The second part of the network is the classification layer, which takes as input the output of the last MaxPool2D layer from the feature generation part. A Flatten layer is then employed to convert the multi-dimensional feature maps into a one-dimensional vector. This vector serves as the input for the subsequent linear and ReLU layers, allowing for further transformation and extraction of relevant information. Finally, the output of the linear layer is passed through another linear layer that redistributes the learned features across the 28 classes, enabling the model to make predictions for each class.

B. Criterion

The criterion chosen is the Cross Entropy Loss, which is a commonly used loss function for classification tasks. It measures the dissimilarity between the predicted class probabilities and the true class labels. The Cross Entropy Loss is well-suited for multi-class classification problems, such as the one in this project, where each input image belongs to one of the 28 classes. Mathematically, the Cross Entropy Loss is defined as follows:

$$\text{CrossEntropyLoss} = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij})$$

where N is the number of samples in the dataset, C is the number of classes, y_{ij} is the ground truth label for the ith sample and jth class (1 if the sample belongs to the class, 0 otherwise), and p_{ij} is the predicted probability for the ith sample belonging to the jth class. The Cross Entropy Loss aims to minimize the discrepancy between the predicted probabilities and the true labels, encouraging the model to assign higher probabilities to the correct classes. By optimizing

this loss function during training, the model learns to make accurate predictions and generalize well to unseen data. In our project, the Cross Entropy Loss is utilized as the criterion for training the Convolutional Neural Network model. It provides an effective measure of the model's performance and guides the learning process towards minimizing the classification error.

C. Activation Function and Optimizer

An activation function is used to introduce non-linearity to the neural network model, enabling it to learn complex patterns and make nonlinear transformations. It applies a mathematical function element-wise to the output of a neuron or a layer. In our model, the chosen activation function is the Rectified Linear Unit (ReLU), which handles negative values by converting them to zero. The ReLU activation function is defined as follows:

$$\text{ReLU}(x) = \max(0, x)$$

In terms of optimization, we chose the Adam optimizer. Adam, an extension of Stochastic Gradient Descent (SDG), adapts learning rates for different parameters, ensuring efficient and effective model optimization. It merges the strengths of AdaGrad and RMSProp, where the dynamic adjustment based on past gradients promotes faster convergence and improved generalization.

D. Model Architecture

We present a detailed visual breakdown of the model and its components, created using the *visualkeras* library.

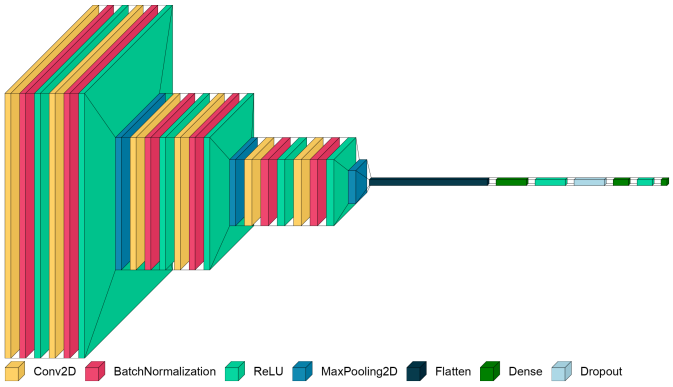


Figure 13. Model architecture

E. Early Stopping

Early stopping is a technique used to prevent underfitting and overfitting of a model. The validation loss is monitored throughout each model epoch using a patience counter, which we have set to 15. During the training, if a model achieves a validation loss lower than the previous epoch, the training continues. Instead, if the model fails to improve after a patience counter of epochs, the training is stopped.

The Early stopping is applied to each model training during hyperparameter tuning process, which will be addressed in the next section, and has been fundamental to reduce training times. To retain the best model for each parameter configuration, we store the current best model to disk. By comparing the parameters based on the best model obtained for each configuration, we can effectively assess and identify the optimal hyperparameters.

The implementation of early stopping not only enhances efficiency but also helps prevent overfitting or premature convergence during the hyperparameter tuning process.

F. GridSearch Hyperparameter tuning

A GridSearch was performed to identify the configuration of parameters that yields the best performance in terms of accuracy and generalization. The goal of the GridSearch is to find the hyperparameter setting combination that gives the best model performance on the validation set. In order to achieve this, a model is trained for every combination of parameters. After evaluating all the parameters, the best model can be selected.

The following hyperparameter values were explored during the HT process:

- **Learning rates**: determine the step size for updating model parameters during training
- **Convolutional units**: represent the size of the filters in the convolutional layer
- **Batch sizes**: determine the number of samples processed in each training iteration
- **Dropout rates**: control the probability of dropping out units during training to prevent overfitting
- **Weight decays**: add a penalty term to the loss function to discourage large weight values

Follows the values explored for each parameter:

- **Learning rates**: [0.0005, 0.0001]
- **Convolutional units**: [16, 32, 64]
- **Batch sizes**: [16, 64, 128]
- **Dropout rates**: [0, 0.2]
- **Weight decays**: [0.01, 0.001]

Due to system memory constraints, the combinations of parameters where convolutional units were set to 64 and batch size was set to 128 were excluded from the grid search. Therefore the total hyperparameter combinations is **56**.

III. Model Elected

A. Best Parameters

In this section, we present the results of our experiments on hyperparameter tuning for our model. During each training, the values monitored were training loss, validation loss, precision, recall, and F1 score. The early stopping procedure also stored the epoch at which the best model was obtained.

From the results, we observe that different hyperparameter configurations yield varying model performance. Models with lower learning rates tend to achieve better validation losses, indicating better generalization. Additionally, increasing the number of convolutional units and using a higher dropout rate generally lead to improved precision, recall, and F1 score.

In the figures 14 and 15 each point of the plot represents the value obtained by training a model with a single parameter configuration of the grid. The best model was chosen by selecting the one which obtained the highest validation F1 score and the lowest loss.

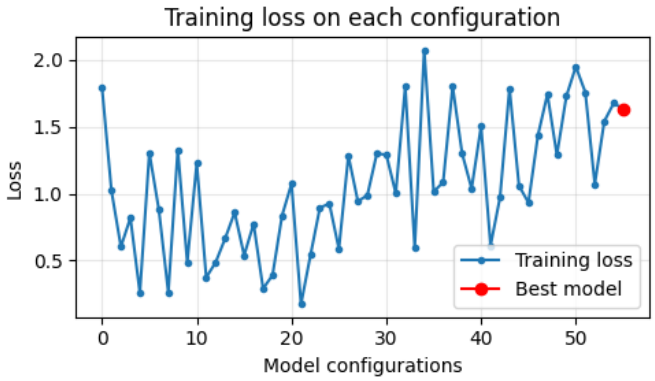


Figure 14. Training loss in each configuration

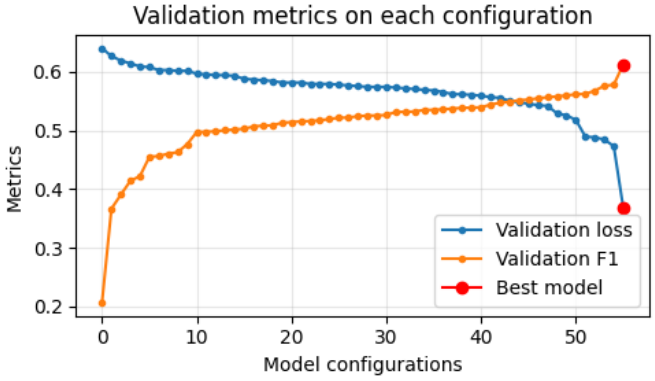


Figure 15. Validation metrics in each configuration

We report the best parameters that allowed to obtain the best model:

- **Batch size**: 16
- **Convolutional units**: 32
- **Dropout rate**: 0
- **Learning rate**: 0.0001
- **Weight decay**: 0.01

The resulting performances of this model are the following:

- **Early stopping epoch**: 98
- **Training loss**: 1.798
- **Validation loss**: 0.640
- **Validation precision**: 0.641
- **Validation recall**: 0.602
- **Validation f1score**: 0.612

In figure 16 it is possible to observe the training history of the best model, where we have monitored the metrics and the early stopping epoch.

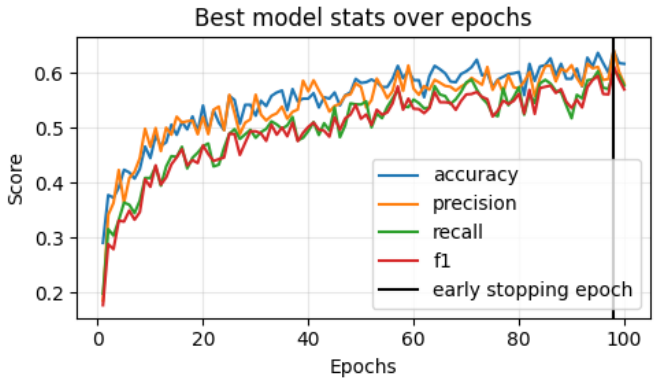


Figure 16. Best model training history

IV. Classification Performance Evaluation

A. Test set

After evaluating the model performance on the validation set, in this section we present the performances relative to the test set. In figure 17 it is reported the confusion matrix relative to the label predicted (on x axis) and the true label (on the y axis). In order to achieve a better visualization, the class labels were removed from the plot, but a summary is reported later in the next paragraph.

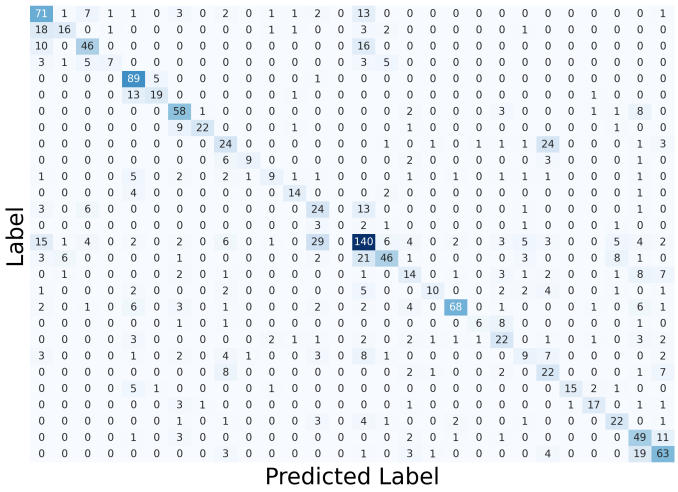


Figure 17. Test set Confusion Matrix

The classification report presents the performance metrics for each class obtained from the Scikit-learn *classification_report* tool. It includes precision, recall, F1-score, and support for each class. The report also provides the results the model's performance for individual classes and overall classification accuracy.

	precision	recall	f1-score	support
bottle-blue	0.55	0.68	0.61	104
bottle-blue-full	0.62	0.37	0.46	43
bottle-blue5l	0.67	0.64	0.65	72
bottle-blue5l-full	0.78	0.29	0.42	24
bottle-dark	0.67	0.94	0.78	95
bottle-dark-full	0.76	0.56	0.64	34
bottle-green	0.64	0.78	0.71	74
bottle-green-full	0.92	0.65	0.76	34
bottle-milk	0.39	0.42	0.41	57
bottle-milk-full	0.82	0.43	0.56	21
bottle-multicolor	0.60	0.32	0.42	28
bottle-multicolorv-full	0.70	0.67	0.68	21
bottle-oil	0.34	0.50	0.40	48
bottle-oil-full	0.00	0.00	0.00	8
bottle-transp	0.60	0.60	0.60	234
bottle-transp-full	0.71	0.50	0.59	92
bottle-yogurt	0.36	0.33	0.35	42
canister	0.71	0.33	0.45	30
cans	0.89	0.69	0.78	98
detergent-box	0.75	0.35	0.48	17
detergent-color	0.47	0.51	0.49	43
detergent-transparent	0.35	0.22	0.27	41
detergent-white	0.31	0.51	0.39	43
glass-dark	0.94	0.60	0.73	25
glass-green	0.74	0.68	0.71	25
glass-transp	0.54	0.61	0.57	36
juice-cardboard	0.46	0.72	0.56	68
milk-cardboard	0.62	0.67	0.64	94
accuracy			0.59	1551
macro avg	0.60	0.52	0.54	1551
weighted avg	0.61	0.59	0.58	1551

The model achieved high precision (0.94) and recall (0.60) for the *glass-dark* class, indicating an accurate classification of positive instances. Classes such as *bottle-blue* (precision: 0.55, recall: 0.68) and *bottle-green* (precision: 0.64, recall: 0.78) also demonstrated good performance with balanced precision and recall values. Some classes had lower precision and recall values, such as *detergent-white* (precision: 0.31, recall: 0.51) and *glass-transp* (precision: 0.54, recall: 0.61).

The class *bottle-oil-full* was never predicted correctly. Its support is low, with just 8 occurrences and by looking at the confusion matrix we observe that the class was majorly misclassified to relative similar classes like *bottle-oil* and *bottle-transp*.

It is worth noting that some classes, such as *bottle-blue* and *bottle-blue-full*, showed slightly lower performance (precision: 0.62, recall: 0.37) due to their similarity, which affected the model's performance.

The overall accuracy of the model on the entire dataset was reported as 0.59.

B. Performance at top-k

In the next section, the precision at top-k is examined. This metric measures the accuracy of the model's predictions when considering the top-k most probable classes. By extracting the first k precisions for each prediction, the best value of k is determined. In figure 18 we report the f1-scores obtained by varying the k value. We can notice that just by increasing the k value to 2, the model performance rise to 0.80.

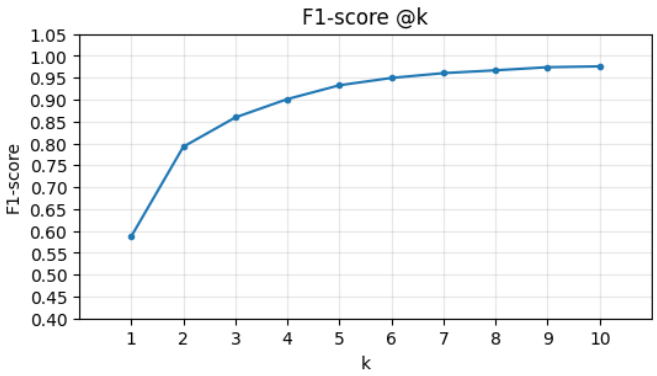


Figure 18. F1-score @k

In this study, a value of $k = 5$ was chosen, which is considered suitable given the total of 28 classes. In the following classification report, we present class precisions, indicating the accuracy of the model's predictions when considering the top 5 most probable classes.

	precision	recall	f1-score	support
bottle-blue	0.99	0.95	0.97	104
bottle-blue-full	0.88	0.84	0.86	43
bottle-blue5l	0.91	0.97	0.94	72
bottle-blue5l-full	0.92	0.96	0.94	24
bottle-dark	0.99	0.99	0.99	95
bottle-dark-full	0.97	0.97	0.97	34
bottle-green	0.99	0.97	0.98	74
bottle-green-full	0.94	0.97	0.96	34
bottle-milk	0.98	0.96	0.97	57
bottle-milk-full	0.90	0.90	0.90	21
bottle-multicolor	0.92	0.82	0.87	28
bottle-multicolorv-full	0.78	0.86	0.82	21
bottle-oil	0.93	0.90	0.91	48
bottle-oil-full	0.44	0.50	0.47	8
bottle-transp	0.98	0.97	0.97	234
bottle-transp-full	0.91	0.90	0.91	92
bottle-yogurt	0.79	0.79	0.79	42
canister	0.71	0.73	0.72	30
cans	0.92	0.92	0.92	98
detergent-box	0.67	0.94	0.78	17
detergent-color	0.98	0.95	0.96	43
detergent-transparent	0.94	0.80	0.87	41
detergent-white	0.84	0.98	0.90	43
glass-dark	0.96	1.00	0.98	25
glass-green	1.00	0.96	0.98	25
glass-transp	0.97	0.86	0.91	36
juice-cardboard	0.93	0.97	0.95	68
milk-cardboard	0.98	0.99	0.98	94
accuracy			0.93	1551
macro avg	0.90	0.90	0.90	1551
weighted avg	0.94	0.93	0.93	1551

Notably, the model demonstrates high precision for several classes, such as *bottle-blue* (.90), *bottle-dark* (.99), and *bottle-transp* (.98). These classes have a high percentage of correct predictions among the top 5 predicted classes. Additionally, the weighted average precision, recall, and F1-score are all above .90, indicating strong overall performance across the classes.

We can notice that the downsampled class *bottle-oil-full* now reaches a precision of .44 and a recall of .50, indicating that the correct class is often returned in the top 5 predictions.

Overall, the results demonstrate the model's ability to provide accurate predictions at the top most probable classes, with a high level of precision and balanced performance across different classes. The overall accuracy of the model is .93.

V. Grad-CAM and Video prediction

A. Grad-CAM

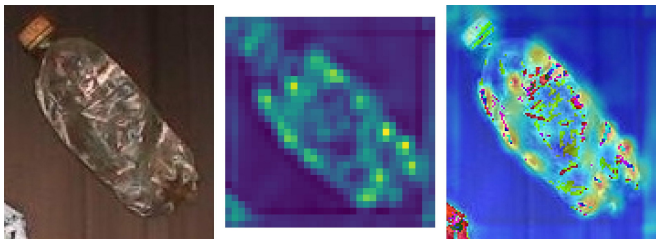
The Grad-CAM technique utilizes gradients between the classification score and the final convolutional feature map to identify the influential regions in an input image. These regions correspond to areas where the classification score heavily relies on the underlying data. The values extracted from the last convolutional layer represent the activations or feature maps obtained after applying the convolutional operation to the input data. These activations capture important spatial information and can be used to understand the learned representations at a deeper level. By analyzing these values, we gain insights into how the model processes and interprets the input data, which can help understanding its decision making process.

The feature map matrix extracted from the last convolutional layer is further processed by scaling and applying colorization techniques. This processed feature map is then overlaid onto the original image, highlighting the specific areas where the activations were detected. By visualizing these activations in the context of the original image, we can gain a better understanding of the regions that contribute most significantly to the model's decision-making process.

In the next section, we'll explore some examples of the Grad-CAM technique applied to a set of images. We'll divide them into two categories: images that were classified correctly and images that were misclassified. By examining the Grad-CAM visualizations for these examples, we can gain insights into why some samples were incorrectly classified and identify the specific regions of the images that played a role in these misclassifications.

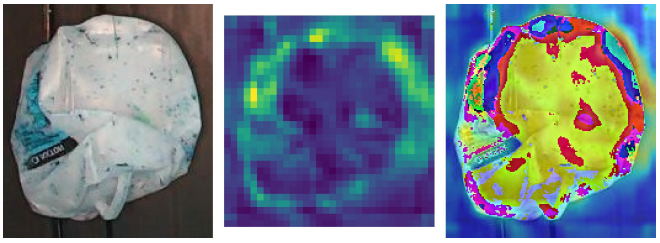
B. Grad-CAM: Correct predictions

True label: *bottle-blue* - **Predicted:** *bottle-blue*



Class	Probability
bottle-blue	0.992
bottle-blue-5l	0.003
bottle-transp	0.002
bottle-oil	0.002
bottle-green	0.001

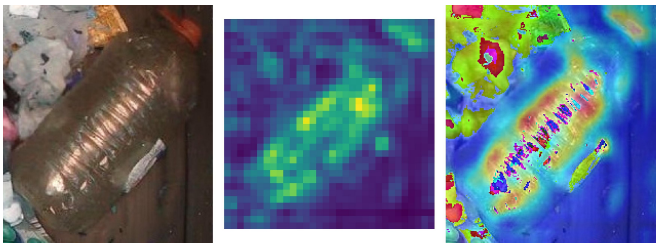
True label: *canister* - **Predicted:** *canister*



Class	Probability
canister	0.805
detergent-white	0.146
bottle-milk	0.045
bottle-multicolor	0.003
detergent-transparent	0.001

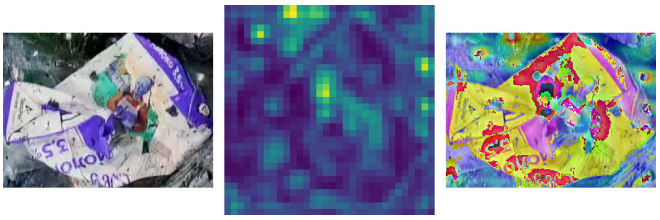
C. Grad-CAM: Incorrect predictions

True label: *bottle-blue5l-full* - **Predicted:** *bottle-blue-full*



Class	Probability
bottle-blue-full	0.625
bottle-blue5l-full	0.368
bottle-blue	0.004
bottle-transp-full	0.003
bottle-green-full	0.001

True label: *milk-cardboard* - **Predicted:** *juice-cardboard*



Class	Probability
juice-cardboard	0.601
milk-cardboard	0.362
detergent-white	0.037
bottle-milk	0.001
bottle-green	¡0.001

D. Real-time prediction on video

In the following section we present the application of the model on videos. We have adapted the model to predict each

frame of a video almost in real time, allowing for efficient and seamless analysis.

To achieve this, we begin by opening the video stream using the *OpenCV* library. We then pass each frame of the video to the trained model for prediction. The model can perform the prediction on images at a high speeds of about 500 images per second, ensuring a low amount of delay in frame display. For each frame we proceed by evaluating the prediction and extracting the top 5 predictions.

For each frame, we do not only display the original image and top 5 predictions, but also visualize the feature map and the Grad-CAM applied to the input image.

To provide a comprehensive view, we combine the original video frame, the top 5 predictions, the feature map, and the Grad-CAM visualization horizontally in an output video. This allows us to observe the original image, the model's predictions, and the corresponding heatmaps simultaneously, facilitating a deeper understanding of the model's performance in a dynamic video context.

We conclude this section by presenting a sample image, in figure 19, that was extracted during the usage of this workflow. This image serves as an example of the real-time prediction process and showcases the capabilities of our approach.

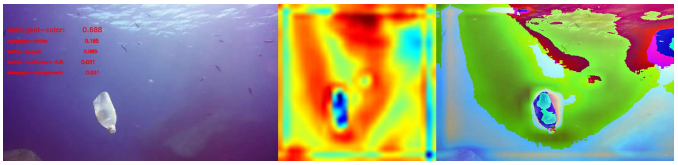


Figure 19. Grad-CAM technique in video streaming

VI. Conclusion

This study utilized the power of artificial vision techniques, specifically Convolutional Neural Networks, to develop a waste image classification model. The model was trained and analyzed using a dataset of recyclable products. We explored various aspects of the dataset, such as image dimensions, class distribution and color normalization.

The data preprocessing and model architecture proved to be effective in learning and extracting relevant features from the input images. The model was trained and evaluated using hyperparameter tuning and early stopping techniques, leading to the selection of the best-performing model configuration.

Furthermore, the Grad-CAM technique was applied to gain insights into the model's decision-making process and identify influential regions in the input images. This visualization technique provided valuable information about the areas of the image that heavily influenced the model's predictions, contributing to a better understanding of the model's behavior.

Lastly, the model was adapted for real-time prediction on video, enabling efficient and seamless analysis. The real-time prediction workflow, along with the visualization of predictions, feature maps, and Grad-CAM, provided a comprehensive view of the model's performance in a dynamic video context.

The developed model showed promising results and can be further improved and applied to real-world scenarios to assist in waste management and recycling efforts.