Food classification on iFood 2019 Challenge

Daniele Cecca
Matr. 914358
MSc Artificial Intelligence for Science and Technology
Email: d.cecca@campus.unimib.it

***Abstract*—This project focuses on the development and implementation of a classification system for food identification. Automatic food identification can assist with food intake monitoring to maintain a healthy diet. Food classification is a challenging problem due to the large number of food categories, high visual similarity between different food categories, as well as the lack of datasets that are large enough for training deep models.**

The project explores two different approaches. In the first one, I use an SVM after the computation of BoW, while in the second approach, I use a custom CNN trained on the original data.

1. Introduction

Today, people are becoming more aware of their health, and food monitoring plays a crucial role in addressing health-related issues. Therefore, systems like this are becoming essential to help individuals follow a correct diet.

This project proposes and compares two different approaches that explore various methodologies. In the first part of the project, I use a Bag-of-Words (BoW) approach with classification performed by a Support Vector Machine (SVM). In the second part, I create a custom Convolutional Neural Network (CNN) trained on the original data.

In both cases, I apply some preprocessing to the data.

The parts are divided into the following sub-steps:

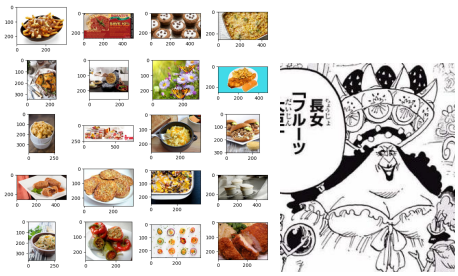
- **BoW with SVM:**
  1) Extract keypoint locations and descriptors using SIFT.
  2) Apply dimensionality reduction using PCA.
  3) Cluster the descriptors to form the vocabulary through quantization, where each cluster represents a word.
  4) Compute the BoW histogram for each image by counting the occurrences of each word.
  5) Classify the images using an SVM model.
  6) Evaluate the model.
- **Custom CNN:**
  1) Create the class dataset.
  2) Design the architecture of the network.
  3) Train the network using different hyperparameters.
  4) Select the best-performing network and train it for additional epochs.
  5) Evaluate the network's performance.

2. Dataset

The dataset is the dataset created for the iFood 2019 Challenge. The training data is composed of 118,475 training images collected from the web while the test data is made of 11,994 images. In total, the dataset contains 251 classes.

With this dataset I have to face two main challenges:

- Fine-grained Classes: The classes are fine-grained and visually similar. For example, the dataset has 15 different types of cakes, and 10 different types of pastas.
- Noisy Data: Since the training images are crawled from the web, they often include images of raw ingredients or processed and packaged food items. This is referred to as cross-domain noise. Further, due to the fine-grained nature of food-categories, a training image may either be incorrectly labeled into a visually similar class or be annotated with with a single label despite having multiple food item



(a) Original images (b) Outlier image

Because finding outliers is outside the scope of this project, I decided not to treat them, even though they could significantly influence the results.

2.1. Dataset Download

To prepare the iFood2019 dataset, the following steps were executed:

- Install Kaggle API: The Kaggle API was installed to facilitate the dataset download.
- Kaggle API Authentication: The Kaggle API was authenticated using a kaggle.json file containing the necessary credentials.
- Download the Dataset: The iFood2019 dataset was downloaded from Kaggle in ZIP format.
- Extract the Dataset: The contents of the ZIP file were extracted to make the dataset accessible for further processing.

2.2. Data pre-processing BoW-SVM

Due to the computational expense of algorithms used to build the vocabulary of features, such as SIFT or K-means, I decided to use only a small portion of the data. The primary limitation was the amount of RAM available in the free version of Colab, which provides only a few GBs. This amount of memory is insufficient to compute and load all the descriptors extracted with SIFT. Thus, subsampling the data was necessary to ensure the feasibility of the computation within the available resources.

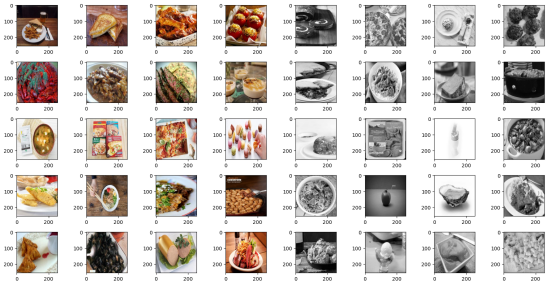
I took 50 samples for each class for training and 10 samples for testing. If some classes don't have 50 samples for training or 10 samples for testing, I decided to sample without replacement. This was the case for class 162.

Then I applied some pre-processing step:

- Resize data
- Convert data to grayscale
- Normalizing pixel values

First of all, I resized the images. In this way, all the images will have the same size. I decided to resize them to 256 x 256 since this was the minimum value among all the images.

Then I converted the images from RGB to grayscale because SIFT only works with grayscale images.



(a) Resize images (b) Gray scale images

And then at the end, I applied histogram equalization using adaptive histogram equalization. In this method, the image is divided into small blocks called "tiles" (the default tile size is 8x8 in OpenCV). Each of these blocks is histogram equalized individually. As a result, in a small area, the histogram will confine to a small region (unless there is noise). If noise is present, it will be amplified.

To avoid this, contrast limiting is applied. If any histogram bin is above the specified contrast limit (the default is 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization. After equalization, bilinear interpolation is applied to remove artifacts at the tile borders.

This method is suitable for improving the local contrast and enhancing the definitions of edges and corners in each region of an image. Thus, it should help the SIFT algorithm to find local features.

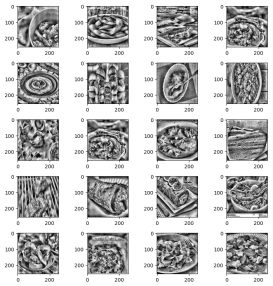


Figure 3: Equalized images

3. Bag of features

The Bag of Features (BoF) model, also known as the Bag of Visual Words, is adapted from natural language processing to represent image features. The process involves three steps:

1) Extract raw features (keypoints and descriptors) from all training images using SIFT.
2) Cluster the descriptors into k clusters using K-means, forming the visual vocabulary.
3) For each image, compute the histogram of visual words by assigning each feature to a cluster and counting occurrences.

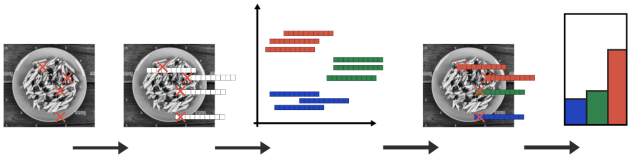


Figure 4: Steps BoW

3.1. Step 1 - SIFT

To extract the features, I used the Scale-Invariant Feature Transform (SIFT) algorithm. SIFT detects distinctive key points or features in an image that are robust to changes in scale, rotation, and affine transformations. It works by identifying key points based on their local intensity extrema and computing descriptors that capture the local image information around those key points

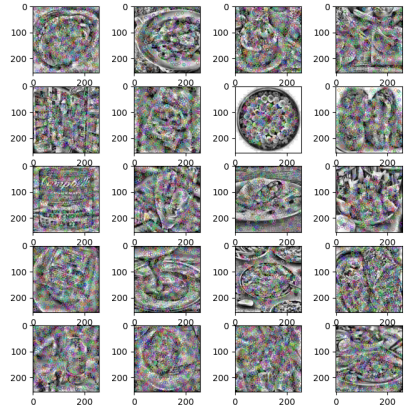


Figure 5: SIFT keypoints

3.1.1. Some consideration on SIFT. By only observing these images I can already say that SIFT captures local features well, but it may miss global contextual information.

3.2. Step 2 - K-means

To determine the number of clusters k, I used the elbow method. Due to computational limits, I could not apply the elbow method to the entire dataset. Therefore, I created a smaller dataset with 10 images per class and then extracted the features using SIFT, resulting in a total of 1268317 features.

Even though it is not an essential step, I decided to apply PCA because the descriptors provide a much sparser set of points to work with. To determine the number of components, I used the Kaiser rule, which involves computing the number of eigenvalues greater than one. In the end, I used 30 components.

I selected as k 750. I chose the 6th iteration because from the graph, the error appears to decrease slowly, making it a good trade-off.

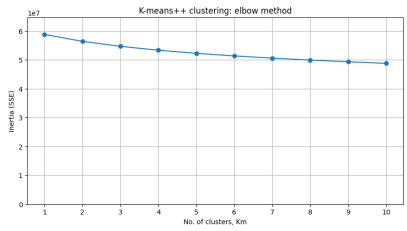


Figure 6: Elbow method

I then computed the visual vocabulary.

3.2.1. Some considerations on BoW. The Bag of Words (BoW) approach simplifies feature vectors, but it loses spatial relationships that are crucial for understanding the overall context of an image.

3.3. Step 3 - Computation of histograms of features

After computing the vocabulary, I generated histograms of features for each image, both for the images in the training set and the test set.

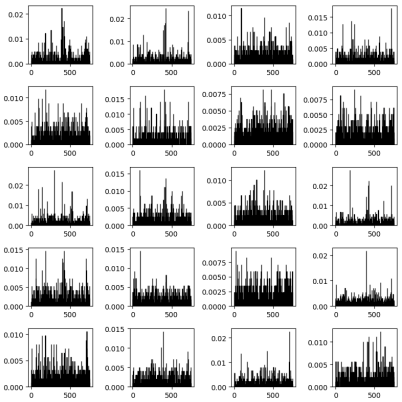


Figure 7: Histograms of features

4. SVM

To classify the images, I utilized the Support Vector Machine (SVM), which is a supervised machine learning algorithm used for classification and regression tasks. SVM finds an optimal hyperplane to separate data points into different classes by maximizing the margin between classes.

To validate our model, I employed Cross-validation. Specifically, I used Stratified Cross-validation with k equal to 3. With Stratified Cross-validation, we ensure that each class in the original training data is represented by the same percentage of samples.

To determine the best hyper-parameters, I defined a grid of parameters and performed a grid search:

Parameter	Values
C	[0.1, 1, 10, 100]
gamma	[1, 0.1, 0.01, 0.001]
kernel	['linear', 'rbf']

TABLE 1: Parameter Grid for SVM Model

where:

- C: The regularization parameter. It controls the trade-off between achieving a low training error and a low testing error (also known as generalization). A smaller value of C encourages a larger margin, at the cost of training accuracy, while a larger value of C aims to classify all training examples correctly.
- gamma: It defines how far the influence of a single training example reaches. Low values of gamma indicate 'far' reach, meaning that each training example influences more points, whereas high values imply 'close' reach, meaning that each training example influences fewer points.

- kernel: Specifies the kernel type to be used in the algorithm. The kernel function determines the shape of the decision boundary.

After the fitting of the different models, the best model has the following hyper-parameters:

Parameter	Value
classifier__C	100
classifier__gamma	1
classifier__kernel	'rbf'

TABLE 2: Best Parameters for SVM Model

5. Evaluation SVM

To evaluate the performance of the model I used the following metrics:

5.1. Some consideration on the evaluation

As I expected the results are very poor among all the metrics.

- **Accuracy:** The ratio of correctly predicted instances to the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** The ratio of correctly predicted positive observations to the total predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** The ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score:** The weighted average of Precision and Recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

And this were the results:

Metric	Value
Accuracy	0.0044
Precision	0.0054
Recall	0.0044
F1 Score	0.0019

TABLE 3: Performance Metrics

I also displayed the confusion matrix.

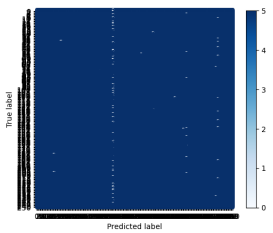


Figure 8: Confusion matrix

5.2. Some consideration about the results

As I expected the results are very poor,among all the metrics:

- The values of accuracy, precision, recall, and F1 score are extremely low (all below 0.01). This indicates that the model's ability to correctly classify instances is severely compromised. In practical terms, this means the model is making incorrect predictions in almost all cases.
- The presence of noise in the dataset significantly impacts the model's performance. Noisy data can result in learning incorrect patterns and associations, thereby impeding the model's capacity to effectively generalize to new, unseen data.
- SIFT-BoW combined with SVM is a traditional approach in computer vision tasks, but its effectiveness can vary depending on the nature of the data and the task. In the case of a noisy and complex dataset, these features and model choice might not be robust enough to handle the noise effectively.
- Using only a small portion of the data might exacerbate the impact of noisy labels. With fewer samples, the model has fewer opportunities to learn meaningful patterns that generalize well.

6. Data pre-processing CNN

A custom dataset class, was created to handle the loading and preprocessing of the images. A DataLoader was used to manage batching and shuffling of the data. From the training set, 10% was extracted as a validation set, while the provided validation set was used as the test set. This approach ensured that the model was evaluated on unseen data during training.

7. CNN architecture

For the second part of the project, I created a custom neural network.

The architecture of this network integrates different types of convolutional layers with residual connections, batch normalization, and adaptive pooling. This combination enhances its learning capabilities and stability while effectively managing the number of parameters. Finally, a fully connected layer is applied for classification purposes.

- **Atrous Convolution (Convolution with dilation):** The network begins with an atrous convolution layer, which allows it to capture information from different parts of the image without losing resolution. This increases the receptive field and enables the network to learn more complex patterns.
- **Inception blocks:** Following the initial convolution, the network employs Inception blocks. These blocks use multiple parallel paths with different filter sizes (1x1, 3x3, and 5x5 convolutions) and a pooling layer. The outputs from these parallel paths are then concatenated. This design allows the network to learn various filter sizes and capture diverse features from the input image.
- **Residual block:** Residual blocks are used to address the shattered gradient problem and to train deeper networks. Each residual block includes:
  - Two convolutional layers
  - Batch normalization layers to stabilize and speed up training
  - A shortcut connection that adds the input of the block to its output
- **Adaptive Max pooling:** After the Inception and Residual blocks, AdaptiveMaxPool2d layers are used to reduce the spatial dimensions of the feature maps to a fixed size. This significantly reduces the number of parameters in the fully connected layers, making the network more efficient and less prone to overfitting.
- **Batch normalization:** After each convolutional layer in the residual blocks, batch normalization is applied. This is crucial for maintaining stable gradient flow and preventing the exploding gradient problem, especially when using residual connections. Additionally, when using a residual network, it is beneficial to invert the order of the functions; therefore, we apply activation to the output.

8. HyperParameters Tuning

To find the best hyperparameters, I experimented with various values across different hyperparameters. Hyperparameters:

- **epochs**: The number of times the entire training dataset is passed through the network.
- **learning rate**: The step size at each iteration while moving towards a minimum of the loss function.
- **drop out**: The fraction of input units to drop during training. Dropout is a regularization technique that helps prevent overfitting by randomly setting a fraction of input units to zero at each update during training.
- **batch size**: The number of training examples utilized in one iteration.
- **loss function**: The function that measures how well the model's predictions match the target values.
- **scheduler lr**: The mechanism for adjusting the learning rate during training, often to improve convergence and performance.
- **optimizer**: The algorithm used to update the weights of the neural network to minimize the loss function.

Specifically, I create a configuration that will be used by the Weights and Biases agent to set the different hyperparameters during various experiments. This allows us to systematically explore the effect of different hyper-parameter settings on the model's performance. In this case, I used the simplest agent which chose the combination of the parameters randomly.

Configuration:

Parameter	Values
epochs	10
learning rate	{0.01, 0.001, 0.0001}
dropout	{0.3, 0.4, 0.5}
batch size	{30, 60, 90}
loss function	CrossEntropyLoss
optimizer	Adam
scheduler	StepLR(optimizer, step_size $= 21, \gamma = 0.2$)

TABLE 4: Hyperparameter Configuration

8.1. Drop-out

Dropout randomly clamps a subset of hidden units to zero at each iteration of computation on the gradient. This makes the network less dependent on any given hidden unit and encourages the weights to have smaller magnitudes so that the change in the function due to the presence or absence of the hidden unit is reduced. This technique has the positive benefit that it can eliminate undesirable "kinks" in the function that are far from the training data and don't affect the loss.

8.2. Adam

Gradient descent with a fixed step size has a notable drawback: it tends to make large adjustments to parameters associated with large gradients, potentially overshooting optimal values, while making smaller adjustments to parameters with smaller gradients, possibly slowing down progress. This imbalance occurs because a single fixed learning rate is not tailored to the varying gradients across different parameters.

To address this issue, I opted to use Adam (Adaptive Moment Estimation), a method that dynamically adjusts the learning rate for each parameter. Adam computes individual adaptive learning rates based on estimates of the first and second moments of the gradients. It maintains two moving averages per parameter, effectively adapting the learning rate over time. This adaptive approach facilitates faster convergence by handling both steep and shallow gradients more effectively, thereby enhancing overall stability and performance of the optimization process.

8.3. Justification for hyperparameter choices

Hyperparameter tuning was limited to the learning rate, dropout rate and batch size due to constraints on model complexity, specifically maintaining a maximum of 1 million parameters. This choice ensured that the model remained within acceptable limits while exploring significant aspects that could impact performance. Additionally, I choose fixed small batch sizes were used because the ADAM optimizer can work more effectively with smaller batches and residual block makes the function more smooth. This approach benefits from more frequent updates, which can lead to better convergence properties and improved generalization. All experiments and training were performed using a T4 GPU.

8.4. Initialization

Initializing the network parameters is critical. To avoid exploding gradient or vanishing gradient, I apply He initialization. The biases β are set to zero, and the weights Ω are chosen from a normal distribution with mean zero and variance $\frac{2}{D_h}$, where D_h is the number of hidden units in the previous layer.

9. Training

The agent run 5 different experiments, with five different combination of hyper-parameters, which are represented in the following picture.

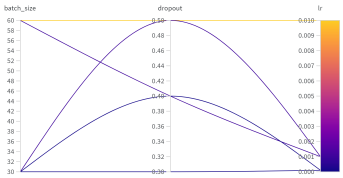
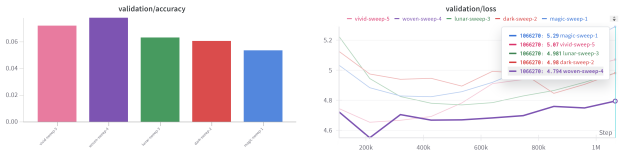


Figure 9: Different combination

I trained all the model with 10 epochs and I evaluate them by computing the validation loss and the accuracy on the validation set.

Figure 10: Train Loss



(a) Accuracy on validation (b) Validation Loss

The model that performed the best was the 4th experiment. Therefore, I chose the 4th experiment as the model for further training, which has the following hyper-parameters:

Parameter	Value
Batch Size	30
Dropout	0.5
Epochs	10
Learning Rate (lr)	0.001

We can observe that the smallest batch size is the most effective, confirming our reasoning about ADAM's performance with smaller batches.

9.1. Training the best model

After training the "best" model for an additional 40 epochs, totaling 50 epochs, I observed the loss function trends. It appears that our model may be overfitting because the training loss continues to decrease while the validation loss increases. This suspicion aligns with the fact that dropout, a highly correlated hyperparameter, tends to improve results as its value increases.

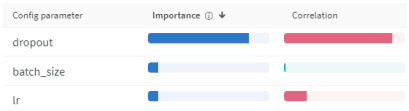


Figure 12: Hyper-parameters importance

Alternatively, the increasing loss could indicate underfitting, where the loss remains within a narrow range due to the model's simplicity, preventing it from capturing complex patterns effectively. Another potential issue could stem from atrous convolutions, which excel in capturing large-scale patterns but may struggle with finer details at a micro-level. Odena et al. (2016) have cautioned that atrous convolution might introduce checkerboard artifacts and should be used cautiously.This challenge could be exacerbated by the residual network, particularly in its initial layers.

Considering these factors, we anticipate poor results from our model.

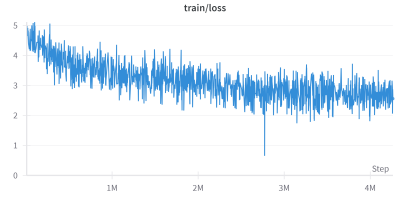
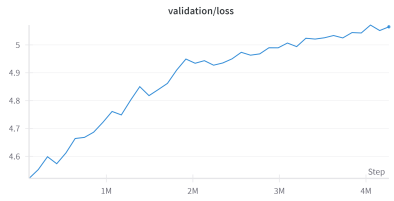
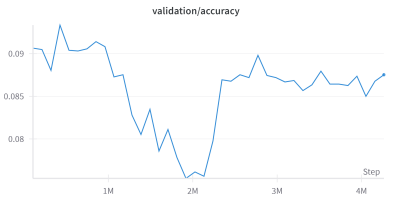


Figure 13: Complete train Loss



(a) Complete val Loss



(b) Complete val accuracy

10. Evaluation CNN

To evaluate the CNN I used the same metrics used for the previous parts:

- Accuracy
- Precision
- Recall
- F1 score

Metric	Value
Accuracy	0.095
Precision	0.086
Recall	0.090
F1 Score	0.084

TABLE 5: Performance Metrics CNN

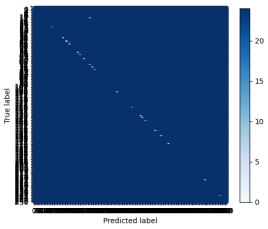


Figure 15: Confusion matrix CNN

10.1. Some consideration about the results

Considering the overfitting symptoms and the challenges with model architecture, it was anticipated that the model might not perform well on new data. The metrics (accuracy, precision, recall, F1 score) confirm these concerns, all hovering around 0.08-0.09, which is indicative of poor predictive power across different evaluation criteria.

Also, as in the case of SVM, noisy data plays an important (in the negative sense) role in prediction.

11. Conclusion

In this project, we explored and compared two methodologies—Bag-of-Words (BoW) with Support Vector Machine (SVM) and a custom Convolutional Neural Network (CNN)—for food image classification using the iFood 2019 Challenge dataset.

The BoW approach initially utilized SIFT for feature extraction, followed by K-means clustering to form a visual vocabulary and SVM for classification. Despite its potential, evaluation revealed limited effectiveness due to the dataset's complexity and noise. Future iterations could benefit from utilizing a larger portion of the original dataset and integrating color descriptors, either through early fusion or late fusion approaches.

Transitioning to CNN architecture, we introduced a custom network tailored for fine-grained food categories and noisy data. This architecture incorporated different convolutional layers, residual connections, batch normalization, and adaptive pooling to enhance feature extraction and classification accuracy. Despite extensive hyperparameter tuning and training strategies, challenges such as overfitting or underfitting persisted.

Looking ahead, future work should prioritize refining data preprocessing techniques to handle noise in fine-grained classifications more effectively. Exploring advanced deep learning architectures specific to food image recognition and try other convolutional layers and hyperparameters could enhance model generalization and efficiency. Additionally, integrating larger and more diverse datasets, possibly curated for food classification tasks, would bolster model robustness.

In conclusion, this project provides foundational insights into food image classification methodologies. However, continued research and development,are crucial to deliver reliable and scalable solutions for real-world applications in health and dietary monitoring.

References

[1] Dataset, *Dataset iFood2019*, https://github.com/karansikka1/iFood_2019?tab=readme-ov-file

[2] Histogram Equalization, *Histogram Equalization*, https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html

[3] SIFT, *SIFT*, https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

[4] scikit-learn SVM, *SVMs*, https://scikit-learn.org/stable/modules/svm.html

[5] scikit-learn K-means, *K-means*, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[6] Understanding deep learning Chap. 7, *He Initialization*, https://udlbook.github.io/udlbook/

[7] Understanding deep learning Chap. 10, *CNN*, https://udlbook.github.io/udlbook/

[8] Understanding deep learning Chap. 11, *Residual*, https://udlbook.github.io/udlbook/

[9] Understanding deep learning Chap. 9, *ADAM*, https://udlbook.github.io/udlbook/

[10] Understanding deep learning Chap 9., *Drop-Out*, https://udlbook.github.io/udlbook/
[11] Weights and biases documentation, *Evaluation CNN*, https://docs.wandb.ai/
[12] A Bag-of-Features Approach based on Hue-SIFT Descriptor, *A Bag-of-Features Approach based on Hue-SIFT Descriptor*, https://www.researchgate.net/publication/257998389_A_Bag-of-Features_Approach_based_on_Hue-SIFT_Descriptor_for_Nude_Detection
[13] Notes of the course of supervised learning, ,

12. Disclosure Statement

The authors declare that this report is entirely original and does not contain any plagiarism.