Comparative Analysis of SIFT/BoW, CNNs, and SSL Techniques for Image Classification

Alessio De Luca
Università degli Studi di Milano-Bicocca
Id. 919790
a.deluca63@campus.unimib.it

Simone Vaccari
Università degli Studi di Milano-Bicocca
Id. 915222
s.vaccari10@campus.unimib.it

Davide Vettore
Università degli Studi di Milano-Bicocca
Id. 868855
d.vettore@campus.unimib.it

***Abstract*—This study compares three methods for food image classification across 251 categories: SIFT with Bag-of-Words (BoW), Convolutional Neural Networks (CNNs), and Self-Supervised Learning (SSL). After addressing the problem of class imbalance, we evaluate the effectiveness of each method, analyzing their strengths and limitations in handling complex visual patterns and class similarities. SIFT struggled the most with distinguishing visually similar classes. The CNN model was designed with a focus on depth and data augmentation, while the SSL approach aimed to leverage features learned from a pretext task. Our analysis provides key insights into the performance of each technique in multi-class classification and identifies areas for future improvement.**

I. Introduction

In this paper, we focus on a supervised learning project based on the iFood-2019 Challenge [1], which aimed to improve food image classification. The challenge was hosted as part of the Fine-Grained Visual Categorization (FGVC) Workshop at CVPR 2019 and involved developing models capable of accurately classifying food images into 251 categories. Food classification is a complex task due to the vast variety of dishes, differences in presentation, and similarities between different food types.

A. The dataset

The original iFood-2019 dataset consists of three subsets: training, validation, and test, each containing images from 251 possible classes. The training set consists of 118,475 samples with labels extracted from the web without verification, which may introduce some noise into the data. The validation set includes 11,994 images, with human-verified labels to ensure higher quality. The test set contains 28,377 images, with non-publicly available human-verified labels used for evaluating the performance of participants' models.

B. The task

The main task consists in classifying food images into one of the 251 categories. To achieve this, we employed three different methods:

- *Feature Extraction Approach:* This method uses SIFT (Scale-Invariant Feature Transform) to extract key points from images. These key points were then quantized into visual words using the Bag of Words (BoW) model. A traditional classifier was then trained over those visual features to perform classification.
- *Convolutional Neural Network (CNN)*: This method leverages deep learning to build a model that learns feature representations from raw images, making it a popular choice for image classification tasks.
- *Self-Supervised Learning:* This approach involves training a similar CNN model on a context-based pretext task to make it learn useful features. These features are then extracted and, along with the labels, used to train a traditional classifier.

II. Data Preparation

For the purpose of this project, we neglected the test set as the ground truths were not available. We began by dividing the training set of 118,475 images into two parts: 70% for training (82,932 images) and 30% for validation (35,543 images). The validation set, which contains 11,994 images with human-verified labels, was used as test set to evaluate model performance.

After splitting the data, we addressed the significant class imbalance present in the training set. With an average of 330 images per class, some classes appeared to be highly underrepresented while others had up to 459 samples, making it necessary to adopt balancing techniques:

- *Downsampling*: For classes with more than 390 images (118% of the average), we randomly removed images until the class count reached the threshold.
- *Upsampling*: For classes with fewer than 264 images (80% of the average), we generated additional images until the threshold was reached. However, instead of simply replicating existing images, we applied various transformations such as rotation, flipping, color jitter, cropping,

and perspective distortion, as shown in Fig. 1. These transformations not only increased the number of images, but also introduced variability, making the dataset more diverse and helping the models generalize better.

After downsampling, the training set decreased to 82,013 images. Following upsampling, the final training set had 83,098 images.

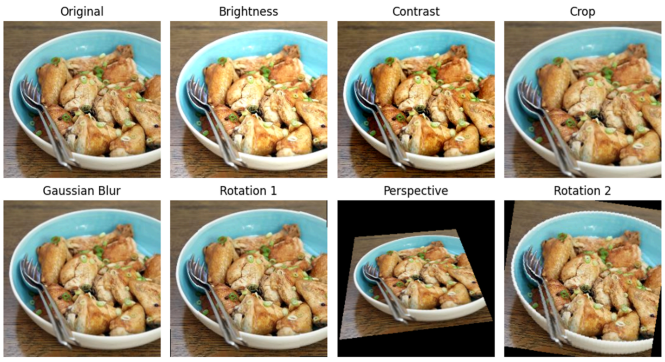

Fig. 1: Transformations applied to introduce variability during upsampling.

III. Image Classification Methodologies

In this chapter, we describe the three methods we used to solve the food image classification task. Each methodology approaches the problem in a different way, extracting and managing features with distinct techniques. We will later compare the models from an evaluational standpoint.

A. *Feature extraction approach*

Scale-Invariant Feature Transform (SIFT) [2] is a robust algorithm used to detect and describe local features in images. It works by identifying keypoints in an image that are invariant to scale, rotation, and even affine transformations. These keypoints are then described using a vector that captures the local image gradients around each point, creating a distinctive feature descriptor. This makes SIFT particularly useful in scenarios where images may vary in terms of orientation, scale, or lighting.

Once SIFT has identified and described the keypoints, the next step involves the Bag of Words (BoW) model, which is a method originally adapted from natural language processing. In the context of image processing, BoW treats image features like words in a document. The keypoint descriptors extracted by SIFT are quantized into a finite vocabulary of visual words through a clustering step, typically using algorithms like K-means. Each image can then be represented as a histogram of these visual words, capturing the distribution of features within the image. This step effectively transforms complex image data into a manageable, fixed-length vector representation that can be used for classification.

Finally, a traditional classifier is trained on these histograms to perform the actual classification task. The classifier learns to distinguish between different categories based on the distribution of visual words within the images. The combination of SIFT, BoW, and a classifier thus forms a powerful pipeline: SIFT ensures that the most informative and stable features are extracted, BoW abstracts these features into a format suitable for machine learning, and the classifier makes the final decision on the class of the image. This method is effective for a wide range of object recognition tasks, particularly when the images have significant variability in appearance and share underlying visual structures.

Our initial approach involved extracting a fixed number of descriptors from all the images in the training set. In a second step, these descriptors were clustered using a K-means algorithm to form the Bag of Features. However, this method had two main limitations. The more intuitive one was the complexity of the K-means algorithm: the number of operations that must be computed can be approximated as $O(n * k * t * d)$ with the variables being respectively the number of descriptors, the number of clusters, the number of iterations needed for convergence, and the dimensionality of the descriptors. Even extracting just 200 descriptors per image from the entire training set would require over 1.5 trillion operations, leading to an estimated computational time of more than 14 hours with a 2GhZ CPU. This led us to switch to Mini Batch K-means, a simpler and less computationally intensive alternative for clustering.

Scikit-learn's `MiniBatchKMeans` [3] reduces computation time by using mini-batches while still optimizing the same objective function. Its `partial_fit` method updates K-means estimates on a single mini-batch. These mini-batches significantly reduce the computation needed to converge to a local solution, producing results only slightly worse than standard K-means.

The second limitation occurred in the descriptor extraction phase: as soon as we started experimenting with more than 200 descriptors per image (a legitimate attempt given that without constraints, up to 3000 descriptors per image were extracted), our 30 GiB RAM failed to handle the load. Particularly, stacking the descriptors at the end, which essentially doubled the RAM usage, caused the system to run out of memory with just over 300 descriptors per image. These two limitations led us to adjust our pipeline.

Since Mini Batch K-means, our most feasible alternative to K-means, already processes descriptors in batches, we decided to merge the two steps. Now, using data loaders, a set number of 1000 descriptors was extracted from the images using `OpenCV` SIFT implementation [4], with the batch size of descriptors different from the one used in the data loaders. The Mini Batch K-means algorithm was then partially fitted on this batch of de-

scriptors. Afterward, the batch was cleared, freeing up RAM, and the process was repeated, each time improving the Mini Batch K-means fit until all descriptors had been processed.

Once the clustering algorithm had been trained, we extracted the descriptors for all images and matched each descriptor to the closest element in the visual dictionary. Each image was then represented as a histogram of these visual words, capturing the distribution of features. Finally, these histograms, combined with the training labels, were used to train a logistic regression classifier.

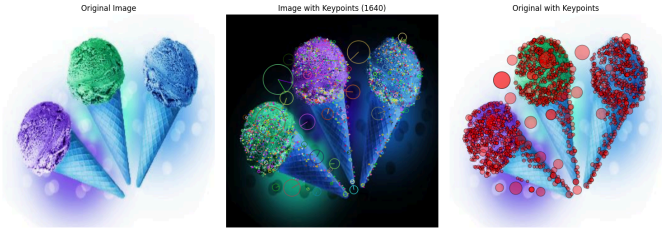

Fig. 2: SIFT keypoint detection on sample image.

The image shown in Fig. 2 demonstrates the application of the SIFT algorithm on a sample image from the training set, which features three ice cream cones. The visualization highlights how SIFT identifies and extracts local features, or keypoints, from the image. In this case, 1640 descriptors have been identified, as we haven't limited them to 1000 in this visualization. The keypoints are represented by circles of different sizes, each corresponding to the scale and orientation of the detected features. Notably, the descriptors are mainly concentrated along the edges of the ice cream scoops and cones, as well as in areas with distinct textures, like the granular surface of the ice cream and the patterns on the cones. These regions contain the most distinctive gradients, which SIFT uses to describe the local features.

B. Convolutional Neural Network

The second methodology applied to tackle the classification task was a Convolutional Neural Network (CNN), a specialized type of artificial neural network particularly suited for computer vision tasks.

The typical architecture of a CNN consists of two main parts:

- Convolutional layers, in which the inputs, the outputs, and the weights are arranged into volumes;
- Fully-connected layers, similar to those in a traditional Multi-Layer Perceptron (MLP), in which the information is organized in the form of a 1-D array. Typically, the last layer works as the predictor;

The convolutional layers progressively extract features from the images, while the fully-connected layers combine these features to generate the final class prediction.

The main challenge of our task was to build a network with fewer than 1 million parameters. For comparison, AlexNet [5], one of the first effective CNNs developed, has approximately 60 million parameters.

The first architecture we designed followed the standard CNN structure: 5 convolutional layers, each with a ReLU activation function and an increasing number of filters, followed by 3 fully-connected layers with a decreasing number of neurons. To reduce the spatial dimensions of the input at each layer, Max Pooling operations were applied after each convolutional layer. A complete summary of this architecture can be visualized in Table I.
Such architecture reached a total of 997,107 parameters.

Layer (type)	Output Shape	Param Num
Conv2d-1	[−1, 8, 256, 256]	608
MaxPool2d-2	[−1, 8, 128, 128]	0
Conv2d-3	[−1, 8, 128, 128]	1,608
MaxPool2d-4	[−1, 8, 64, 64]	0
Conv2d-5	[−1, 16, 64, 64]	1,168
MaxPool2d-6	[−1, 16, 32, 32]	0
Conv2d-7	[−1, 16, 32, 32]	2,320
MaxPool2d-8	[−1, 16, 16, 16]	0
Conv2d-9	[−1, 32, 16, 16]	4,640
MaxPool2d-10	[−1, 32, 8, 8]	0
Linear-11	[−1, 400]	819,600
Dropout-12	[−1, 400]	0
Linear-13	[−1, 256]	102,656
Dropout-14	[−1, 256]	0
Linear-15	[−1, 251]	64,507

TABLE I: SUMMARY OF THE ARCHITECTURE OF THE FIRST CNN.

The results obtained with this model were not very satisfactory. We assumed this was due to the limited number of convolutional layers and filters applied, imposed by the constraint on the number of parameters of the network. Therefore, after several attempts to organize the layers in a better way while not exceeding one million parameters, we decided to employ a different strategy.

Inspired by the Xception network [6] and the MobileNets [7], the second model we developed was based on depthwise separable convolutions.

Depthwise separable convolutions are a type of convolutional operation designed to reduce the computational cost and the number of parameters of a CNN by breaking down the standard convolution operation into two simpler and more efficient steps: depthwise convolution and pointwise convolution.

- *Depthwise Convolution:* unlike standard convolutions that use K×K× C_{in} filters, where K is the size of the kernel and C_{in} is the number of input channels, with each filter spanning all input channels, depthwise convolution applies a single filter per input channel. This means each input channel has its own K×K filter, resulting in C_{in} output channels.
- *Pointwise Convolution:* a simple 1×1 convolution is used to create a linear combination of the output of the depthwise layer.

In practice, if we have an input of size M×M× C_{in}, to produce an output of size M×M× C_{out} a standard convolutional layer would apply C_{out} K×K× C_{in} filters. On the other hand, depthwise convolution applies one K×K filter to each input channel, resulting in a volume of size K×K× C_{in}. Then, pointwise convolution would apply C_{out} 1×1× C_{in} kernels to obtain the desired output volume.

Table II shows the difference in computational cost and number of parameters in the two cases.

	Standard Convolution	**Depthwise Separable Convolution**
Computational Cost	$M \times M \times C_{in} \times C_{out} \times K^2$	$M \times M \times C_{in} \times (K^2 + C_{out})$
Number of Parameters	$C_{in} \times C_{out} \times K^2$	$C_{in} \times K^2 + C_{in} \times C_{out}$

TABLE II: STANDARD CONVOLUTIONS VS DSC.

The reduction in terms of number of parameters of the network is remarkable: for example, with $K = 3$, if we had 64 input channels and 128 output channels, the standard convolution would require 73,7K parameters, while depthwise separable convolution would need only about 8,7K parameters.

The other new feature we introduced was a Global Average Pooling (GAP) layer following the depthwise separable convolution blocks. Global Average Pooling is a pooling operation designed to reduce the spatial dimensions of a feature map by computing the average of all values in that feature map. Specifically, when applied to a volume, GAP takes each feature map in the input and computes the average of all spatial elements, resulting in a single value per feature map. Therefore, if the input feature map is of size M×M×C, after applying GAP, the output will be 1×1×C, which can be flattened to a C-dimensional vector. This method of feature aggregation allows us to replace fully-connected layers, leading to a significant reduction of the number of parameters.

After tuning the number of layers and filters, and other hyperparameters, such as Dropout, Batch Normalization, and Max Pooling layers, the final configuration of our network was the one presented in Table III.

Layer (type)	**Output Shape**	**Param Num**
Conv2d-1	[−1, 32, 256, 256]	2,432
BatchNorm2d-2	[−1, 32, 256, 256]	64
Conv2d-3	[−1, 32, 256, 256]	9,248
BatchNorm2d-4	[−1, 32, 256, 256]	64
MaxPool2d-5	[−1, 32, 128, 128]	0
Conv2d-6	[−1, 32, 128, 128]	320
Conv2d-7	[−1, 64, 128, 128]	2,112
BatchNorm2d-8	[−1, 64, 128, 128]	128
Conv2d-9	[−1, 64, 128, 128]	640
Conv2d-10	[−1, 128, 128, 128]	8,320
BatchNorm2d-11	[−1, 128, 128, 128]	256
MaxPool2d-12	[−1, 128, 64, 64]	0
Conv2d-13	[−1, 128, 64, 64]	1,280
Conv2d-14	[−1, 256, 64, 64]	33,024
BatchNorm2d-15	[−1, 256, 64, 64]	512
Conv2d-16	[−1, 256, 64, 64]	2,560
Conv2d-17	[−1, 512, 64, 64]	131,584
BatchNorm2d-18	[−1, 512, 64, 64]	1,024
MaxPool2d-19	[−1, 512, 32, 32]	0
Conv2d-20	[−1, 512, 32, 32]	5,120
Conv2d-21	[−1, 1024, 32, 32]	525,312
BatchNorm2d-22	[−1, 1024, 32, 32]	2,048
MaxPool2d-23	[−1, 1024, 16, 16]	0
AdaptiveAvgPool2d-24	[−1, 1024, 1, 1]	0
Dropout-25	[−1, 1024]	0
Linear-26	[−1, 251]	257,275

TABLE III: SUMMARY OF THE ARCHITECTURE OF THE SECOND CNN.

This resulted in a total of 983,323 trainable parameters, respecting the requirement while allowing for a greater number of convolutional layers.

Before starting the training routine, the images were preprocessed to ensure they were in a suitable format for our network. In particular, each image was resized to a standard size of 256×256 pixels, and an augmentation procedure was applied when loading the images of the training set. Data augmentation is a technique commonly used in training machine learning models, especially for image-based tasks. The main purpose of data augmentation is to artificially increase the size of the training dataset by creating modified versions of the original images. This helps the model generalize better to new, unseen data by introducing variability and reducing overfitting.

In our project, we employed the Albumentations library for data augmentation [8], implementing the following transformations:

- Horizontal Flip: Randomly flips the image horizontally with a probability of 0.5. This helps the model learn that objects can appear in different horizontal orientations.
- Vertical Flip: Randomly flips the image vertically with a probability of 0.5, adding another layer of variability.
- Random Rotation: Rotates the image by a random angle in the range [−30, 30] with a probability of 0.5. This ensures the model can handle different rotations of objects.
- Color Jitter: Changes the brightness, contrast, saturation, and hue of the image with a probability of 0.5. This operation helps the model become more robust to different environments, lighting conditions, and color distortions that might occur in real-world scenarios.

The images were then normalized, using the normalization parameter values from ImageNet.

While resizing and normalization were applied to all the images, the augmentation operations were only applied to the training samples.

Fig. 3 shows examples of preprocessed training images with their corresponding ground truth labels. The applied normalization enhances the images for better model interpretation. Based on the ground truth labels, we know that the dishes are *french fries*, *fruitcake*, and *ziti*, a typical shape of pasta.

Fig. 3: Sample of preprocessed training images with ground truth labels.

The training routine was performed using the Cross Entropy loss function, which for a batch of size N can be defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} I(y_i = j)\log(p_{ij})$$

where C is the number of classes, y_i is the true label of the i-th sample, p_{ij} is the predicted probability for sample i belonging to class j, and

$$I(y_i = j) = \begin{cases} 1 \text{ if } y_i{=}j \\ 0 \text{ othewise} \end{cases}$$

In practice, Pytorch `CrossEntropyLoss` class [9] doesn't even require to compute the probabilities, but works with raw logits. The model was trained multiple times for a different number of epochs, and an early stopping criterion was defined: if the validation loss does not improve after 5 epochs, the training is interrupted. Training was conducted with the Adam optimizer and a learning rate scheduler. The latter monitors the validation loss and halves the learning rate after 3 epochs of non-improvement, allowing the model to adjust its learning parameters before training is stopped.

Validation was performed after every epoch. During these phases, we kept track of the trend of the loss, along with the total accuracy.

C. *Self-Supervised Learning*

Self-Supervised Learning (SSL) has emerged as a powerful approach in scenarios where labeled data is scarce or expensive to obtain. Traditional supervised learning tasks, like training CNNs on large datasets, require a significant amount of labeled data. Labeling such large datasets is often time-consuming and costly. SSL addresses this challenge by using the data itself to generate pretext tasks, enabling models to learn useful features without requiring explicit labels for every image. These features can then be extracted or fine-tuned for specific tasks, such as image classification, reducing the need for large amounts of labeled data and potentially improving model generalization.

Several types of SSL methods exist, each designed to learn from data in different ways by solving different pretext tasks:

- *Generation-based methods:* These involve training models to produce new data, like images or text, from an initial input. Examples include Generative Adversarial Networks (GANs), which focus on generating or reconstructing data samples.
- *Free semantic label-based methods:* These approaches use labels like segmentation masks, depth images, and optical flows, which are generated automatically without human annotation. These labels can be rendered by game engines, such as Airsim [10] or Carla [11], or created using hard-coded methods.
- *Cross-modal methods:* In cross-modal SSL, the model learns by predicting the relationship between different types of data, such as images and text. This approach allows the model to understand and link multiple modalities of data.

In our project, we focused on a *Context-Based* SSL method, specifically using a spatial context structure known as a Jigsaw Puzzle. We chose this approach because it allows the model to learn spatial relationships between different parts of an image, which is crucial for detailed tasks like food classification. Among the context-based methods, the Jigsaw Puzzle task is particularly effective for our dataset as it helps the model capture local features and understand the spatial arrangement of food items.

The goal of a Jigsaw Puzzle pretext task is to rearrange shuffled patches of an image into their correct positions, similar to solving a real Jigsaw Puzzle. For our implementation, we di-

vided each image into a 4x4 grid, resulting in 16 patches. The patches were then randomly shuffled, and the model was asked to predict the correct positions of the shuffled patches. The number of possible permutations of these 16 pieces is given by 16!, exceeding 20.9 trillion. While some implementations limit the number of permutations using Hamming distances, our model achieved remarkable performance in the pretext task, suggesting the task wasn't overly ambiguous and meaningful features were extracted.

The architecture used for this task is based on the CNN model previously described in the section above. However, it has been modified to process images with a 4x4 shuffle. Several convolutional layers capture features from the input images, and a fully-connected classification module maps these learned features to predict the position of each shuffled image patch.

The preprocessing steps used for the traditional CNN, such as data augmentation, were also included in this implementation to add variability to the dataset, which remains beneficial. The training loop was similarly defined for consistency: it used a Cross Entropy loss function and employed an early stopping criterion. The same optimizer and scheduler were used across both approaches.

After training was completed, we used the learned features for the main classification task. These features were extracted from the Global Average Pooling layer preceding the final classification layer. Extracting them from earlier layers, like the first two convolutional layers, would result in a much higher number of features compared to deeper layers. The number of elements produced by each layer forms a map of dimension $\text{batch size} * \text{channels} * \text{height} * \text{width}$. If we extracted features from an early convolutional layer, we would end up with over two million attributes per image, which would be impractical. The features, along with their labels, were then used to train a logistic regression classifier, which was responsible for categorizing the images into their respective food categories.

IV. Results

In this section, we will present the results obtained from the three methodologies applied in the study. We will first discuss the evaluation metrics used, followed by a detailed analysis of the performance of each model.

To evaluate the effectiveness of the models in the multi-class classification task, we employed several metrics:

- *Accuracy*: It's the overall percentage of correctly classified images out of the total number of images. This metric reflects the proportion of correct predictions across all classes.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

- *Precision*: For each class, this metric computes the ratio between true positive predictions and the sum of true positive and false positive predictions. The final value represents the weighted average over all classes.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

- *Recall*: This metric is the ratio between the true positive predictions and the sum of true positive and false negative predictions for each class.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

- *F1-score*: The F1-score is the harmonic mean of precision and recall, which provides a single measure of the model's performance.

$$\text{F1-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

In particular, we will primarily focus on accuracy. This metric is the most relevant for our study since it reflects the overall effectiveness of the model in correctly classifying all classes uniformly.

A. *Models Evaluation*

The Scale-Invariant Feature Transform paired with the Bag of Words model was the only pipeline not based on deep learning, but instead relied on feature extraction and traditional classification methods. The main parameters we had to tune were the maximum number of descriptors extracted from each image and the size of the visual vocabulary. For computational reasons, the number of extracted descriptors needed to be limited, and we did this by fixing the `nfeatures` parameter to 1000. Alternatively, we could have filtered out features by adjusting the contrast and edge thresholds. Overall, while experimenting with the first approach, we noticed that extracting a higher number of features improved results but also increased the complexity of the clustering process proportionally.

We also had to tune the number of clusters, which effectively form the visual vocabulary. We experimented with 200, 500, and 1000 clusters, and found that beyond a certain point, increasing the vocabulary size no longer improved performance but only slowed down the process. As a result, we settled on 500 clusters. More rigorous methods for finding the optimal K in the K-means algorithm include heuristics like the elbow method. This involves performing clustering over a range of K, then plotting the within-cluster sum of squares and looking for an elbow-like dip. However, since feature extraction and clustering were performed together in batches in the final pipeline, repeating this process for a wide range of K values was very time-consuming.

Finally, the extracted BoW histograms were scaled using z-score normalization to speed up convergence and improve the results. Paired with the labels from the training set, a simple logistic regression model was trained. The resulting performance metrics obtained over the test set are reported in Table IV.

	Accuracy	Precision	Recall	F1
SIFT+BoW	6.26%	5.77%	6.25%	5.87%

TABLE IV: FEATURE EXTRACTION APPROACH TEST PERFORMANCE METRICS.

Since this approach doesn't rely on learning through a deep neural network, one might expect it to be less time-consuming. However, extracting and clustering such a large number of descriptors per image makes it almost as slow as training a CNN for 30 epochs, while achieving less than half the performance. Once the features are extracted, training a linear classifiers takes only a few minutes.

Fig. 4 shows the first 9 out of 500 visual words from the vocabulary. Since visual words represent cluster centers, the corresponding patches were obtained by identifying the closest SIFT descriptors to each cluster center across a batch of images. We could have used color-coded patches, but since SIFT doesn't utilize color information, we decided to keep the visual representation consistent with grayscale data.

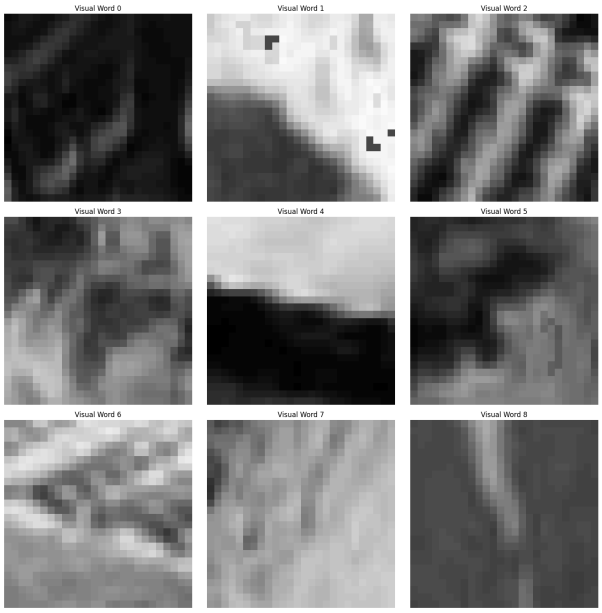

Fig. 4: Visualization of representative image patches for 9 visual words.

Better results were achieved using our deep learning models. We began by training a very simple and small model, consisting of just a couple of convolutional layers and very few linear layers, to ensure it met the constraint on the number of parameters. As expected, this network showed poor performances from the very first epochs. The configuration presented in Table I was achieved after careful refinement. In particular, we aimed to keep the feature volume at the end of the convolutional part of the network as small as possible, while still deep enough to prevent excessive information loss. This was necessary to limit the number of parameters in the first linear layer, which contains the majority of the network's parameters. To achieve this, we reduced the spatial dimensions of the feature maps after each convolutional layer through pooling operations, allowing us to create a deeper network. The number of neurons of the second and third linear layers was then set to the highest possible value that kept the parameter count below one million, following the common approach of using progressively smaller layer sizes. This architecture was trained for a maximum of 30 epochs with batches of 32 images. Training was completed after 1 hour and 44 minutes, without early stopping. The trained CNN was able to achieve nearly 12% accuracy on the test set, an improvement with respect to the previous results, though still not satisfactory.

The approach followed for the second CNN (Table III) allowed us to increase the network's depth, while avoiding an excessive reduction in the dimension of the feature volume, ensuring an appropriate level of model complexity. After being trained for 30 epochs, this network reached 30,66% accuracy on the test set, confirming the effectiveness of depthwise separable convolutions for small models. As this resulted to be our best-performing model, we decided to train it for 50 epochs, after noticing a linear trend in the learning curves, indicating that the network had not overfitted. Fig. 5 and Fig. 6 show the trends of the training and validation loss, as well as training and validation accuracy, respectively. It can be noted how even after 50 epochs the validation loss continues to decrease with an almost monotonic trend, suggesting that the network is still learning and could benefit from additional training epochs. However, considering the time-consuming nature of the training process and the limited computational resources at our disposal, we decided not to proceed further.

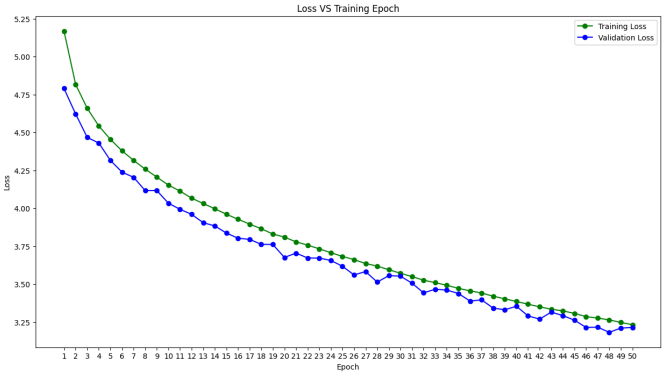

Fig. 5: Training (green) and validation (blue) loss trend.

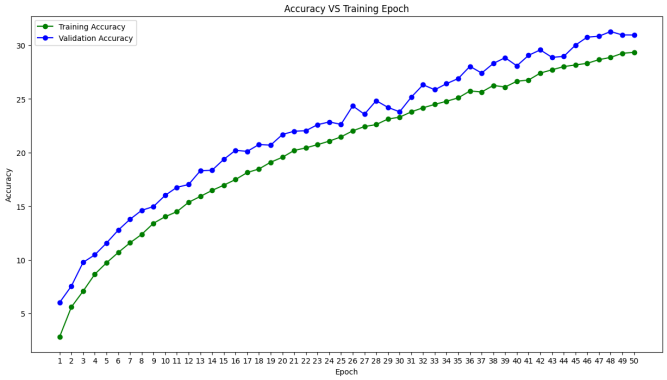

Fig. 6: Training (green) and validation (blue) accuracy trend.

Table V reports the values of the metrics of interest scored by our trained models on the test set. The total training time for the 50-epoch run was around 8 hours.

	Accuracy	**Precision**	**Recall**	**F1**
CNN-1 30 epochs	11.98%	9.53%	11.98%	8.96%
CNN-2 30 epochs	30.66%	33.90%	30.66%	28.86%
CNN-2 50 epochs	36.84%	39.13%	36.84%	35.98%

TABLE V: CNN TEST PERFORMANCE METRICS.

After reaching a satisfactory result, we analyzed some of the predictions made by the network. Some examples can be visualized in Fig. 7.

Fig. 7: Predicted and Ground Truth labels for some test samples.

In the first case, the network correctly predicted the test sample as *apple_turnover*. The second image, which shows *chicken_marengo*, was misclassified as *lobster_bisque*, a soup dish containing pieces of lobster. Finally, in the third case, the *mostaccioli* dish was labelled as *manicotti*. While this example reflects the network's accuracy (on average, 1 over 3 images is correctly classified), it also highlights that most misclassifications are due to high similarities among the classes in the dataset. For instance, there are over 15 different types of pasta, 6 types of cakes, and 5 kinds of soup dishes.

The Self-Supervised Learning approach was built upon the best-performing CNN architecture and refined for the pretext task. We initially trained the model for 25 epochs. The learning curves for the Jigsaw Puzzle task are shown in Fig. 8. Additionally, we tested the model's performance after 3, 5, and 10 epochs. This choice was driven by the fact that our main interest isn't in how well the model solves the pretext task, but rather in its ability to extract meaningful features for the main classification task. As we will explore later, these two objectives are not directly correlated.

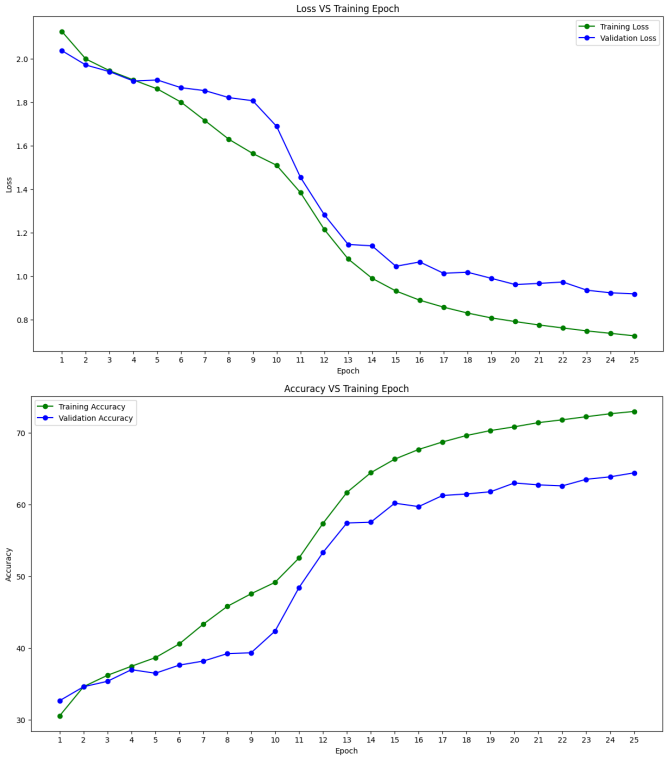

Fig. 8: Training (green) and validation (blue) loss and accuracy trends.

The features extracted from the last layer of the network trained on the pretext task were used to train a linear classifier. The features for each image were normalized and stored along with their respective labels. Table VI shows the performance of models trained for different epochs, both on the pretext task and the image classification task.

	Accuracy *Pretext task*	**Accuracy** *Classification task*	**Precision**	**Recall**	**F1**
3 Epochs	36.19%	9.06%	9.11%	9.06%	8.94%
5 Epochs	37.84%	**9.52%**	9.67%	9.52%	9.41%
10 Epochs	45.40%	7.90%	7.88%	7.90%	7.74%
25 Epochs	**65.88%**	3.94%	3.75%	3.94%	3.75%

TABLE VI: SSL TEST PERFORMANCE METRICS.

The results demonstrate that superior performance in the Jigsaw Puzzle task does not necessarily lead to better classification results. This is because the Jigsaw Puzzle tests the model's ability to solve a specific image arrangement problem, which is different from classification. Surprisingly, models with shorter training times, which performed worse on the Jigsaw Puzzle task, actually provided better features for classification. This is likely because they didn't overspecialized on the pretext task and thus retained more general and useful feature representations.

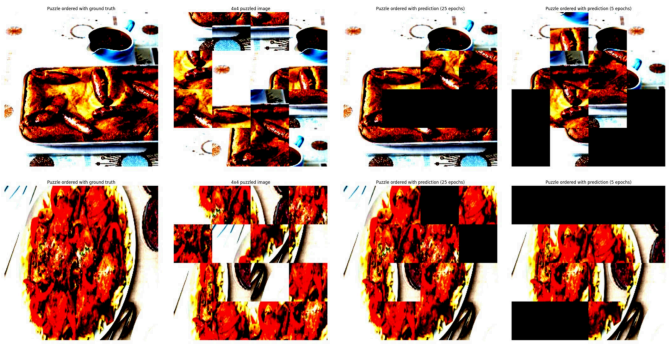

Fig. 9: Comparison of predicted patch orders by models trained for 25 and 5 epochs.

Fig. 9 shows two samples of input images processed by the Self-Supervised Learning CNN. The original image is presented, along with the 4x4 puzzle generated by the network, and the corresponding order predictions made by models trained for 25 and 5 epochs, respectively.

The black spaces are not plotting errors; they occur because the network occasionally assigns the same index to multiple patches, leaving some positions unfilled. As the model goes through the puzzled patches, it predicts where each patch belongs, but it can overwrite previously assigned positions, leading to missing areas. To prevent this, the network would need to be more complex, ensuring that each position is used only once, either by preventing overwrites or allowing for revision of previous predictions. A possible improvement could be a slight modification in the architecture to incorporate a permutation-based prediction strategy, which would likely improve performance. However, as we've already concluded, we're not concerned with achieving high performance on the pretext task. In fact, we are intentionally using the features extracted by the 5-epoch model for the image classification task, despite its weaker performance on the Jigsaw Puzzle.

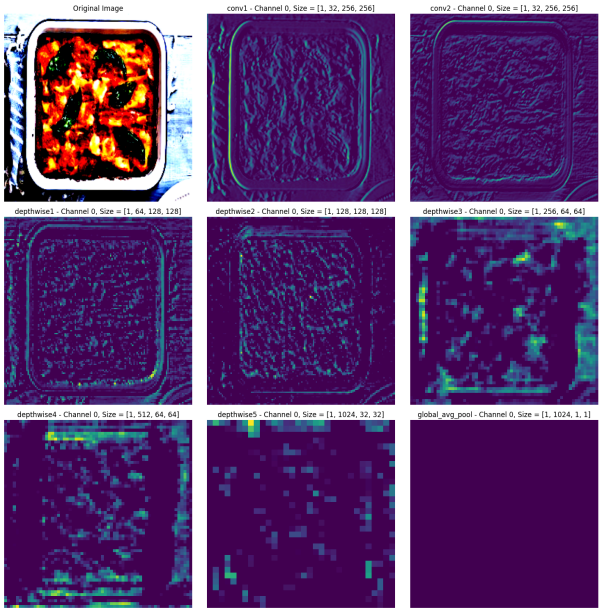

Fig. 10: Feature maps evolving from detailed to abstract layers.

Fig. 10 illustrates the feature maps extracted from different stages of the CNN as it processes an input image, starting with the original image in the top left and progressing through various layers. In each of the layers, the network computes multiple feature maps, with the number increasing as we go deeper into the network. For example, conv1 has 32 filters, but only the first feature map is visualized here for simplicity.

In the initial convolutional layers (conv1, conv2), the feature maps capture low-level details such as edges and textures. These early-stage feature maps still retain some recognizable structure of the input image. As we move deeper into the network, through the depthwise separable convolution layers (depthwise1 to depthwise5), the feature maps become progressively more abstract, capturing higher-level concepts.

Finally, the global average pooling layer condenses the extracted features into a single $1\times1\times1024$ feature map, summarizing the most important information from all previous layers. While this final representation no longer resembles the input image, it efficiently encodes the essential details used for the classification task.

V. Conclusions

This study aimed to compare the performances of a feature extraction-based approach and a convolutional neural network, both in supervised and self-supervised frameworks, for food image classification.

In the feature extraction approach, we used a SIFT model to capture keypoints and descriptors at various scales. While this method has been proven to excel in instance recognition,

it struggled with the visual similarities across our 251-class dataset. Many of these classes share similar visual features, and SIFT failed to capture sufficiently discriminative information. For this reason, despite the computational expense and fine-tuning of parameters, the classification results were limited, achieving only 6.26% accuracy.

In contrast, the CNN, despite having fewer than 1 million parameters, performed well due to its ability to capture complex patterns for image classification. Its depth enabled the extraction of both low-level and abstract features, which proved crucial for distinguishing between visually similar food categories. Data augmentation further enhanced the CNN's performance by introducing variability into the training data, improving generalization. This technique wasn't used in the SIFT approach since we were only extracting descriptors, and augmentations could have introduced noise or altered keypoints. Overall, the CNN shows the best balance between computational cost and accuracy, achieving 36.84% accuracy.

Usually, a Self-Supervised Learning approach consists in using the weights from the network trained on the pretext task to initialize the main CNN, and then fine-tune it reducing the need for large labeled datasets. In our case, however, we extracted the learned features and applied a traditional classifier to make predictions. Despite using the same architecture, the different classification approach with SSL led to inferior outcomes with respect to the CNN, achieving only 9.52% accuracy. One notable finding was that shorter training epochs in the pretext task resulted in better classification features. This suggests that over-specialization might limit the model's ability to generalize for the main classification task.

Finally, let's consider some potential improvements. In the feature extraction approach, exploring alternative detection algorithms could be useful. For instance, Binary Robust Independent Elementary Features (BRIEF) is a feature descriptor that, when combined with an appropriate feature detector, is less computationally demanding and might offer better recognition rates. There are also SIFT variants that include color information, which could enhance performance. However, a major improvement could be shifting to a feature matching-based approach rather than sticking with a traditional classifier. This would transform the problem into an instance recognition task, where these algorithms generally perform better, compared to standard classification. Lastly, we could have tried different traditional classifiers, but we chose to stick with a linear one because of its simplicity in handling high-dimensional data and its quick convergence.

In the deep learning approach, experimenting with larger architectures and advanced techniques like residual connections or deeper separable convolutions could lead to further improvements. Adjusting data augmentation strategies to introduce more variability might also enhance performance. Exploring top-5 accuracy could provide insight into whether the model's predictions were semantically close to the correct class. Alternatively, it could be useful to develop a performance metric that weights errors differently based on whether the prediction is a completely different food or, for example, just a different pasta shape.

The SSL approach offered an interesting perspective by using a context-based task to extract features. However, a key improvement would be to use the approach as it was meant to be used: this means fine-tuning networks instead of training separate classifiers. Additionally, experimenting with alternative pretext tasks or different layers for feature extraction may yield better results.

We confirm that this report is entirely original and free from plagiarism. The research were conducted exclusively by the authors, without the use of any automated language models. All sources we used are properly cited in the references.

References

[1] P. Kaur, , K. Sikka, W. Wang, s. Belongie, and A. Divakaran, "FoodX-251: A Dataset for Fine-grained Food Classification," *arXiv preprint arXiv:1907.06167*, 2019.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[3] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., Curran Associates, Inc., 2012, p. . [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[6] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions." [Online]. Available: https://arxiv.org/abs/1610.02357

[7] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." [Online]. Available: https://arxiv.org/abs/1704.04861

[8] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," *Information*, vol. 11, no. 2, 2020, doi: 10.3390/info11020125.

[9] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035, 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[10] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles." [Online]. Available: https://arxiv.org/abs/1705.05065

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator." [Online]. Available: https://arxiv.org/abs/1711.03938