Supervised Learning project report

Aly Valiev
Student ID: 911861
University of Milano-Bicocca, Milan
Email: a.valiev@campus.unimib.it

***Abstract*—This project explores image classification using both traditional and deep learning approaches. We preprocess images. Feature extraction is performed using ORB, followed by classification with Random Forests on the BoW representations. Additionally, a custom Convolutional Neural Network (CNN) is trained for comparison. We evaluate model performance using accuracy, precision, recall, and F1-score, and conduct hyperparameter tuning to optimize results.**

1. Introduction and dataset description

Food classification is a challenging problem due to the large number of categories, high visual similarity between different foods. Here we explore this task with FoodX-251 dataset with Fine-grained Classes and Noisy Data [1].

1.1. Dataset exploration

This dataset contains 251 fine-grained food categories with 118,475 training images collected from the web. Human verified labels are provided for the validation set of 11,994, which is used as a test set in this report. The initial set of 118,475 was split into training and validation sets.

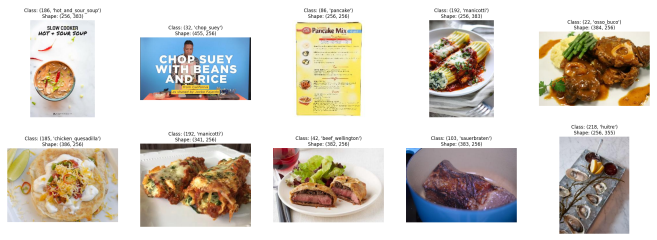

Figure 1: Example of images.

As we can see above the dataset is noisy and may be difficut to classify even for a person. Some images classification is obviously wrong, as in the Figure 2.

The class distribution is imbalanced, as we can se in the Figure 3. Each bar correspond to one of the food classes.

Overall, the classification task on this dataset presents a significant challenge due to the noisy data, evident misclassifications, and imbalanced class distribution.

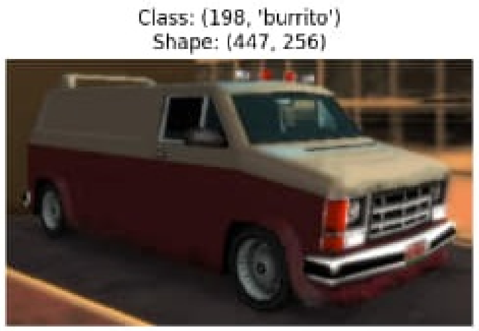

Figure 2: Example of the bad data.

2. Traditional Machine Learning Approach

2.1. BoW representation

First we used the ORB (Oriented FAST and Rotated BRIEF) algorithm for feature extraction. Then we applied KMeans clustering to generate a visual vocabulary. Each image got the Bag of Words histogram representation, which was then used to train a Random Forest classifier.

Images were reshaped to the same size (200 by 200 pixels) and converted to gray scale. ORB was chosen for feature extraction due to its computational efficiency and robustness. The speed of the algorithm was a crucial factor. due to the large size of the dataset, with 251 classes. For each image maximum of 100 features were calculated.

To get visual vocabulary we used the MiniBatchKMeans algorithm, a variant of KMeans suited for large datasets. We set the number of clusters to 2500. The algorithm clustered the feature descriptors into 2500 distinct groups, each representing a visual word in the BoW model.

Each image had a number of extracted descriptors. The trained KMeans model was used to predict the cluster assignments for each descriptor. It allowed us to compute a BoW representation with histogram, showing the frequency of each visual word within the image. Example of a BoW representation is shown in the Figure 4.

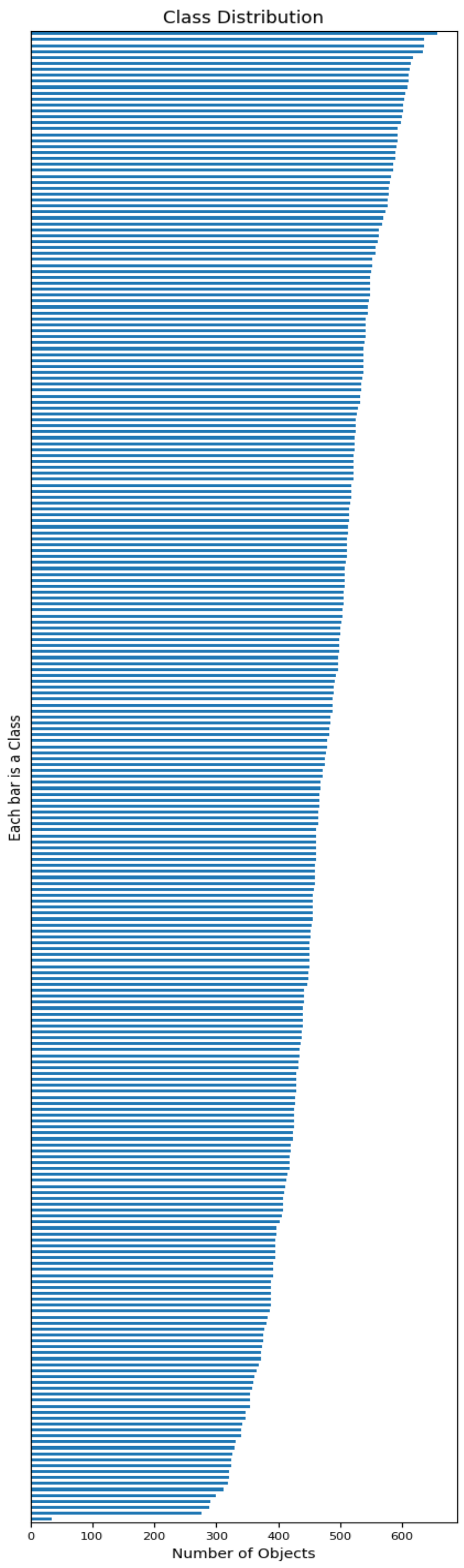

Figure 3: Class distribution.

2.2. Random Forest Classifier and hyperparameter tuning

Then traditional Random Forest Classifier was trained on BoW representation calculated before.

We conducted a grid search. This involved testing various combinations of `'max_depth'` and `'n_estimators'`. Each combination was evaluated by training the model on the training set and testing its performance on the validation set. The model with the highest validation score was selected as the best.

The best set of hyperparameters was taken and new model was trained on a combined training and validation sets. Then the model was finally evaluated on the test set, that was not used before.

Dataset	Accuracy	Precision	Recall	F1-score
Training Set	0.0409	0.4596	0.0409	0.0526
Test Set	0.0082	0.0047	0.0082	0.0020

TABLE 1: Performance metrics for the training and test sets.

The table 3 shows the performance metrics for the training and test sets of the model. Model performs very poorly, additionally we can see the substantial drop in performance from the training set to the test set.

As noted above, this dataset is a challenging one, especially for traditional machine learning algorithms. There are several potential improvements that could be explored to enhance model performance, including:

- Higher number of features in the ORB algorithm, allowing the detection of more detailed and distinctive characteristics of the images.
- Increasing the number of clusters in the KMeans algorithm, expanding the visual vocabulary.
- Testing other traditional classifiers.

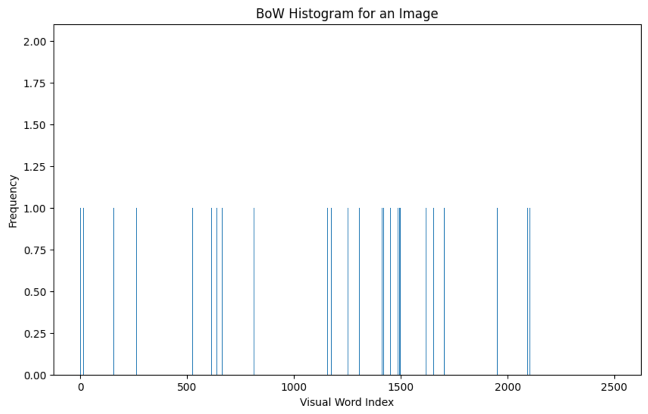

Figure 4: Example of the BoW histogramm.

3. CNN approach

Given the challenges faced by traditional machine learning algorithms with this dataset, we tried to train Convolutional Neural Networks (CNNs), which have shown remarkable success in image classification tasks.

In this section, we will describe the architecture and training process of a CNN. We will detail the model's structure and present the results.

Additionally, we had a constraint of a maximum of 1 million parameters, which significantly limited our ability to explore more complex and potentially more effective model architectures.

3.1. Simple CNN

First we preproccesed images. Each image was resized to 100x100 pixels (we made images smaller to avoid exceeding the parameter limitations due to the dense layers of the network). The resized images were converted to PyTorch tensors and pixel values were normalized.

Our first CNN consists of three convolutional layers followed by max-pooling layers and two fully connected layers. The network's architecture and number of parameters can be seen below in the Table 2. As observed, despite the significant reduction in image dimensions after the convolution and max-pooling layers, the linear layer still demands a substantial number of parameters. This constraint limits our ability to add additional convolutional layers and increase the model's complexity.

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 4, 100, 100]	112
Conv2d-2	[-1, 8, 50, 50]	296
Conv2d-3	[-1, 16, 25, 25]	1,168
Linear-4	[-1, 256]	590,080
Linear-5	[-1, 251]	64,507
Total params:	**656,163**	

TABLE 2: Simple CNN Model Summary

The model was trained using the Adam optimizer with learning rate of 0.001 and CrossEntropyLoss. Early stopping was implemented with a patience of 5 epochs to prevent overfitting. If the validation loss did not improve for 5 consecutive epochs, training was stopped early.

This model perfomed even worse than a traditional classifier.

Dataset	**Accuracy**	**Precision**	**Recall**	**F1-score**
Training Set	0.0040	0.0001	0.0040	0.0002
Validation Set	0.0037	0.0001	0.0037	0.0002
Test Set	0.0053	0.0001	0.0053	0.0002

TABLE 3: Performance metrics of the simple CNN model.

Additionally most of the classes were not predicted at all. We need a more complex architecture for a model to actually learn.

3.2. Enhanced CNN

To improve the model's performance while keeping the parameter count below 1 million, we defined a new CNN model with a different architecture, as shown in Table 4. We took inspiration from the MobileNet architecture [2]. The use of global average pooling replaces the need for a large fully connected layer, allowing us to reduce the parameter count and add additional convolution layers. The last convolutional layer uses depthwise separable convolutions. This method splits the convolution operation into a depthwise convolution and a pointwise convolution, reducing the number of parameters compared to a traditional convolutional layer. Additionally, batch normalization and dropout are applied after each convolutional layer to stabilize the training and avoid potential overfitting.

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 32, 200, 200]	896
BatchNorm2d-2	[-1, 32, 200, 200]	64
Dropout-3	[-1, 32, 100, 100]	0
Conv2d-4	[-1, 64, 100, 100]	18,496
BatchNorm2d-5	[-1, 64, 100, 100]	128
Dropout-6	[-1, 64, 50, 50]	0
Conv2d-7	[-1, 128, 50, 50]	73,856
BatchNorm2d-8	[-1, 128, 50, 50]	256
Dropout-9	[-1, 128, 25, 25]	0
Conv2d-10	[-1, 256, 25, 25]	295,168
BatchNorm2d-11	[-1, 256, 25, 25]	512
Dropout-12	[-1, 256, 12, 12]	0
Conv2d-13	[-1, 256, 12, 12]	2,560
Conv2d-14	[-1, 512, 12, 12]	131,584
BatchNorm2d-15	[-1, 512, 12, 12]	1,024
Dropout-16	[-1, 512, 6, 6]	0
AdaptiveAvgPool2d-17	[-1, 512, 1, 1]	0
Linear-18	[-1, 512]	262,656
Dropout-19	[-1, 512]	0
Linear-20	[-1, 251]	128,763
Total params:	**915,963**	

TABLE 4: Model summary of the EnhancedCNN architecture.

The training logic is the same as for the training of a Simple CNN with early stopping implemented.

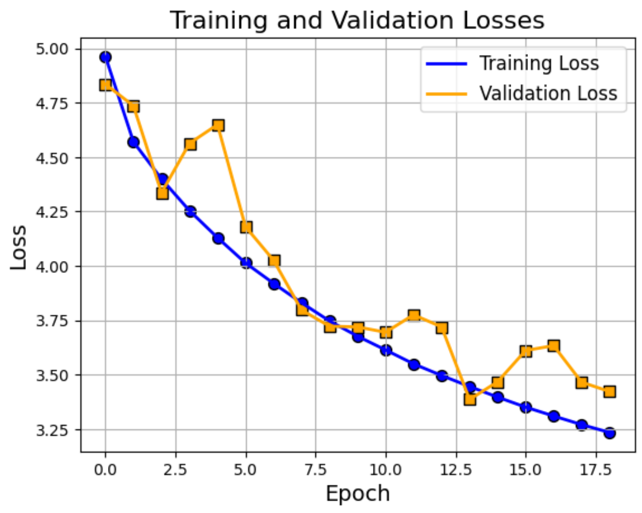

Figure 5: Training and Validation Losses.

3.2.1. Results. More complex CNN's performance showed a significant improvement compared to the previous models.

Only a few classes were not predicted at all:

- Class: 65 - scallop
- Class: 243 - mostaccioli
- Class: 238 - rugulah

Below the performance metrics are shown. Additionally, a top-5 accuracy was calculated with a value of 0.5785, meaning that more than half of the images had their correct class within the top 5 predictions.

Metric	**Value**
Accuracy	0.2908
Precision	0.3197
Recall	0.2908
F1-score	0.2696
Top-5 Accuracy	0.5785

TABLE 5: Performance metrics for the test set.

Unfortunately, the random split between the training and validation sets was lost because the random seed was not specified. The current model took approximately 7-8 hours to train using Kaggle's GPU accelerator. Due to the long training time, we were not able to retrain the model immediately. However, we plan to retrain it and will publish the performance metrics for the training and validation sets on the project's GitHub.

There is still a lot of room for improvement. Using outlier detection techniques to find wrong or mislabeled images can lead to cleaner training data. Additionally, experimenting with different dropout rates and batch sizes could help optimize the model's performance. Another possibility is to make the model even more complex by incorporating residual blocks, L2 normalization, and more depthwise separable convolutions.

4. Conclusion

In this project, we explored image classification using traditional machine learning and CNN approaches on the challenging FoodX-251 dataset. The traditional approach with ORB for feature extraction and Random Forests for classification based on Bag of Words representations. Despite hyperparameter tuning, the traditional model struggled to achieve satisfactory performance.

Further we implemented and trained CNNs. Simple CNN performance was even worse than the that of traditional approaches. Then we developed an enhanced CNN inspired by MobileNet, incorporating global average pooling and depthwise separable convolutions to maintain a balance between complexity and parameter count. This enhanced model demonstrated a significant improvement in performance metrics.

The enhanced CNN achieved a better results than previous models, showing model's ability to handle fine-grained classification tasks better than traditional methods, although there is still considerable room for improvement

Acknowledgments

The authors declare that this report is entirely original and does not contain any plagiarism. The research explained in this report was conducted by the authors themselves, and all the sources have been cited in the Reference Section. None of the content was generated using automated language models.

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] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.