Food Image Classification using CNNs

Benedetta Bensi
Artificial Intelligence for Science and Technology
Università degli studi di Milano-Bicocca
867143
Email: b.bensi@campus.unimib.it

Stefano Carotti
Artificial Intelligence for Science and Technology
Università degli studi di Milano-Bicocca
911255
Email: s.carotti1@campus.unimib.it

***Abstract*—Image classification is a central task in computer vision, the objective is to assign a label to an image from a predefined set of categories. This report explores the application of CNNs in both supervised and self-supervised learning paradigms.**

1. Introduction

Supervised learning refers to learning methods using data with human annotated labels to train the networks, meaning each training example is paired with an output label. The goal is to make the model predict the corresponding labels, matching the human annotated ones. If labels are missing or partially missing or even if the data annotation is too expensive in terms of time or resources, supervised learning becomes unfeasible, making self-supervised learning a valid alternative. Self-supervised learning indicates learning methods that exploit large amounts of unlabeled data to create supervisory signals, which can reduce the need for labeled data and improve the model's ability to generalize.
Convolutional Neural Networks (CNNs) are a class of deep neural networks primarily used for analyzing visual data. They are especially effective for image classification and object detection. They are inspired by the visual cortex structure in animals, where neurons respond selectively to specific features. In this report we developed a computational model smart enough to perform automatic food identification.

2. Dataset description and preprocessing

Food classification poses significant challenges due to the high variety of food categories and the visual similarity that can occur between different types of food. The dataset at our disposal presented 251 fine-grained food categories with training and validation images collected from the web. In particular the 118,475 training images were divided with a percentage of $80\% - 20\%$, to compose the training and validation splits and the provided validation dateset was utilized as test set, resulting in the following numbers:
Number of training samples: 94784
Number of validation samples: 23695
Number of test samples: 11993

More in detail, from 100 to 600 images per class were present and some classes are visually similar. For example, the dataset has 15 different types of cakes, and 10 different types of pastas [1]. In the following figure are presented two similar pasta dishes corresponding to different labels:

(a) Spaghetti carbonara

(b) Spaghetti bolognese

Figure 1. Raw images from the train set.

As can be noticed the input size of the images is uncontrolled, therefore a first resizing step is needed in order to make the network accept them. In particular, we decided to resize to a dimension of 256×256. In addition to this, it was checked that all the images were in RGB scale since the inputs of our CNN requires the three channels. Another observable thing is that some images were taken from various angles. In order to have a uniform distribution in the training set, normalization was applied by computing the channel-wise mean and standard deviation of the pixel values in the dataset. For consistency we used the same mean and standard deviation to normalize both the validation and the test set.

Figure 2. Normalized image.

3. Supervised Learning

Supervised learning is a machine learning method where the algorithm is trained on labeled data. In this approach, each training example includes input features and a corresponding target output. The aim is to learn a mapping from inputs to outputs, so that it can predict the label for new, unseen data.

The process involves three main steps: training, validation and testing. During the training phase, the algorithm processes the training data and adjusts its parameters to minimize the loss between its predictions and the actual targets, during the training the model is fed with unseen data (validation data) to provide an unbiased evaluation of the model performance and choose the best model and hyper-parameters. Once trained, the model performance is evaluated using a separate test dataset to verify whether it generalizes well to new data.

3.1. Custom Convolutional Neural Network

Convolutional layers apply a set of filters (kernels) across the input image to produce features, each filter slides across the entire image, performing element wise multiplication followed by a sum (convolution), this way features like colours, edges or corners are found. The filter uses the same set of weights across all the different positions in the image. A stack of convolutional layers helps in abstracting the features and capturing complex structures within the data. Then for each convolved value in the feature map resulting from the layer, a non-linear activation function is applied, like the Rectified Linear Unit (ReLU). The activation function is usually followed by a pooling layer, which reduces the dimension of the feature map while preserving important information. The convolutional part of the network retrieves the features that are used by the fully connected classifier to predict the label. Here the spatial dimension is destroyed with the input of the first fully connected layer being a one dimensional array.

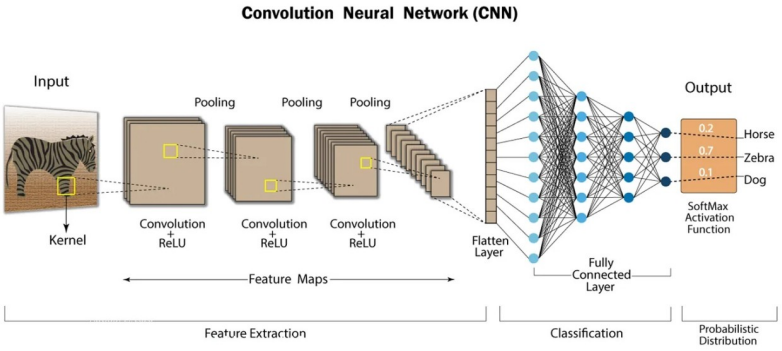


Figure 3. Typical architecture of a Convolutional Neural Network.

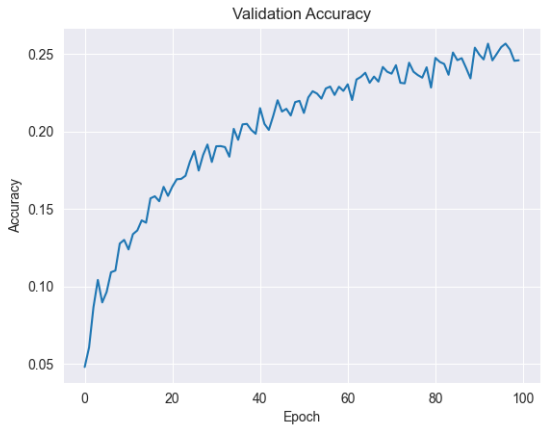
Once the image is fed to the network it produces a probabilistic distribution of the existing classes, this output is evaluated by a loss function, in our multi-class setting cross-entropy was utilized, that measures the error between the output and the ground truth label. The backward-propagation is the process of computing gradients of the loss with respect to the parameters of the network so that these parameters can be changed in the direction that minimizes the error.
Since our task was to build a custom CNN with less that one million parameters, we started by defining a simpler model:

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 16, 126, 126]	1,744
MaxPool2d-2	[-1, 16, 42, 42]	0
Conv2d-3	[-1, 32, 37, 37]	18,464
MaxPool2d-4	[-1, 32, 18, 18]	0
Conv2d-5	[-1, 32, 13, 13]	36,896
MaxPool2d-6	[-1, 32, 6, 6]	0
Linear-7	[-1, 440]	507,320
BatchNorm1d-8	[-1, 440]	880
Linear-9	[-1, 251]	110,691
Total params: 675,995		
Trainable params: 675,995		
Non-trainable params: 0		

In particular we tested average and max pooling, but noticed that the second got better results. We started using as optimizer Stochastic Gradient Descent (SGD), but then switched to Adaptive Moment Estimation (Adam) with learning rate equal to 0.001. At first the model was trained for 20 epochs, but through the plots of training and validation loss and accuracy we have seen that the model was still improving steadily, therefore we increased the epochs up to 100. Early stopping was implemented with a *patience* value of six, meaning that the validation accuracy had to worsen for six consecutive epochs to stop the training. This was in order to prevent overfitting and save time.

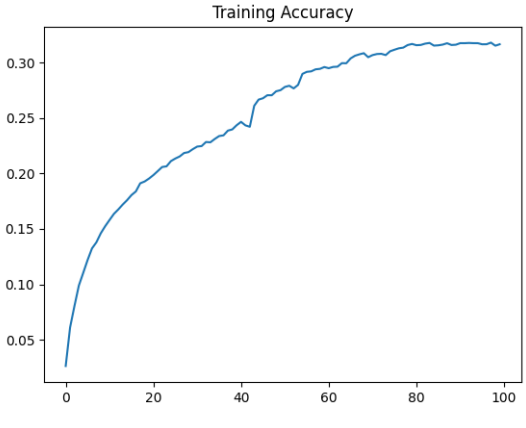
A validation accuracy of 24.6% was achieved. Keep in mind that, although low, this accuracy is still promising,

(a)

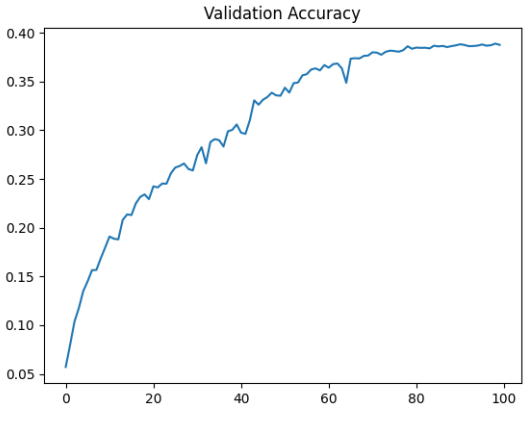


(b)

Figure 4. First model results.



(a)



(b)

Figure 5. First model results with LR scheduler.

since the task was to perform classification on 251 classes and the baseline accuracy for a random guess is 0.398%. The plots in figure 4 indicates that there is potential for further improvement by increasing the number of epochs, but this was not pursued for computational reasons.
At this point we decided to implement a learning rate scheduler in order to improve the accuracy of the model. *ReduceLROnPlateau* was chosen, which reduces the learning rate when the training stagnates. The hyper-parameters were the factor by which the learning rate would be reduced (0.5) and the patience which defines how many epochs without improving in the validation accuracy were needed for the scheduler to act (2 epochs).

With this setting, around 40% of accuracy was achieved in the validation set. We checked how it behaved on the test set and were surprised by seeing approximately 17% of accuracy. Such a discrepancy meant overfitting: due to the scheduler that acted on the learning rate only accordingly to the validation set, the network adapted to this set and as a consequence we had an overestimation on its generalization performances on unseen data. To try to prevent this problem a step scheduler was applied (0.5 factor every 15 epochs) together with a dropout layer after the first fully connected (50%). Since we had similar results to Figure 4 and no more overfitting, we decided to make the model more complex by increasing the number of parameters and increase the depth of the convolutional part.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 252, 252]	1,216
MaxPool2d-2	[-1, 16, 126, 126]	0
Conv2d-3	[-1, 32, 122, 122]	12,832
MaxPool2d-4	[-1, 32, 61, 61]	0
Conv2d-5	[-1, 64, 59, 59]	18,496
MaxPool2d-6	[-1, 64, 29, 29]	0
Conv2d-7	[-1, 128, 27, 27]	73,856
MaxPool2d-8	[-1, 128, 13, 13]	0
Conv2d-9	[-1, 56, 11, 11]	64,568
MaxPool2d-10	[-1, 56, 5, 5]	0
Linear-11	[-1, 500]	700,500
BatchNorm1d-12	[-1, 500]	1,000
Linear-13	[-1, 251]	125,751
Total params: 998,219		
Trainable params: 998,219		
Non-trainable params: 0		

We implemented a learning rate of 0.0004 as it showed the best results both in terms of accuracy and preventing overfitting among all the trials. By setting 70 epochs with early stopping, an accuracy of 31% was obtained on the validation set and 30.3% on the test set.
Other models and learning rates were tried in order to achieve a better accuracy, but none of them overcame the test set accuracy of 30%. Nevertheless, it is still a good result considering the size of the task, the number of epochs, the constraint on the number of parameters on our model and the poor quality of the training set which will be investigated in the later sections.

3.2. Best model evaluation

Since we had our best model, we evaluated its performance on a separate test dataset. The metrics utilized were accuracy, precision, recall and F1-score. Precision for a given class in multi-class classification is the fraction of instances correctly classified as belonging to a specific class out of all instances the model predicted to belong to that class. Recall in multi-class classification is the fraction of instances in a class that the model correctly classified out of all instances in that class. In order to take account of imbalanced classes, the weighted average of precision and recall was computed, which calculates metrics for each label, and find their average weighted by support (the number of true instances for each label). The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0 [2].
Accuracy: 30.3%
Precision: 0.308
Recall: 0.303
F1 Score: 0.287

3.3. Investigation of the results

For a detailed analysis of the classification results and to have an insight into where the model is making errors in order to allow for specific improvements, we decided to utilize the confusion matrix. The confusion matrix is an $n \times n$ matrix where each row represents the actual class and each column represents the predicted class. The diagonal elements of this matrix represent the number of instances correctly classified for each class. In particular,

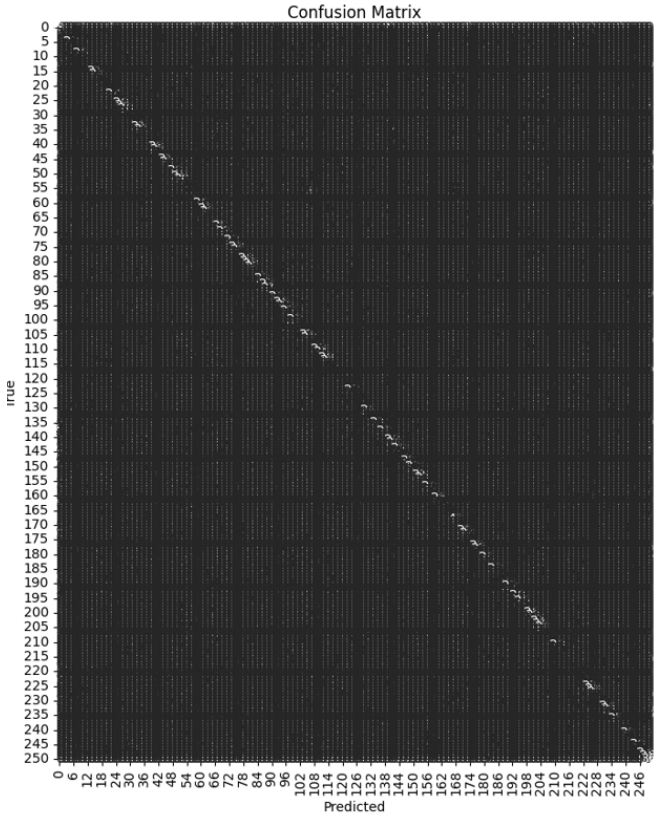


Figure 6. Confusion matrix best model.

we looked for the lowest values on the diagonal, meaning the classes with the worst classification accuracy. Among them, class 103 was present, corresponding to *sauerbraten* dish. We decided to inspect this class and noticed that this is one that has fewest training images and that it was mainly misclassified with class 231 (*carbonnade flamande*). We found that they are both stews from different countries and appear very similar visually. Additionally, class 231 is more prevalent in the training set, making it reasonable to believe that class 231 dominates over the other in training. Another class that had the worst classification results was class 116, corresponding to *poi* dish. Inspecting among the training images, a vast majority of outliers was found. More specifically, all these outliers were images of the same category. We searched on the web what *poi* was and figure it out that it is both a dish and a theatrical prop. Since the

images of this dataset are samples from the web, we think this is the reason why this class has so many outliers and, as a consequence was completely misclassified. A similar problem occurs in class 58, *bubble and squeak* dish, to which also a children cartoon is associated. By a quick inspection among the worst classified we found even more classes with a vast majority of outliers. To try out whether

(a) Class 116 outlier sample.

(b) Class 58 outlier sample.

Figure 7. Outliers from the train set.

these outliers have an impact on our model, we tried to exclude only classes 116 and 58 from train, validation and test sets and re-trained our best model.
Accuracy: 30.9%
Precision: 0.305
Recall: 0.309
F1 Score: 0.3

Even though we cannot make any statistically founded conclusion based on these results, it was the first time we had an improvement, even if little, on most of these metrics. Take in consideration that by taking out the negative examples the accuracy was expected to increased, but by having less false positives also the learning of the model has improved in quality. Since the outliers were the majority of the training instances in these classes, by deleting such classes the network didn't have to learn patterns such as edges or shapes of completely different objects compared to the real purpose of the network classification. We think that by better refining and taking out only the outliers instead of the whole classes and further investigating on the remaining classes, the results could be enhanced in a significant way.

4. Self-Supervised Learning

Traditional supervised learning relies on large amounts of labeled data, which can be expensive and time-consuming to obtain. The core idea behind self-supervised learning is to design pretext tasks for the network to solve. These are tasks in which pseudo-labels can be extracted directly from the data "for free" allowing the network to be trained by minimizing a given objective functions. As a consequence, solving these tasks helps the model learn useful representations of the data that often capture the underlying patterns and structures and extract features that can be transferred to downstream tasks, as in our specific scenario, image classification.

Images contain rich spatial context information such as the relative positions among different patches from an image, which can be used to design the pretext task for SSL. We used *Jigsaw puzzle* pretext task, where an image is divided into nine non-overlapping patches that are then shuffled. The shuffled image patches are fed into the network which is trained to recognise the correct spatial locations of the input patches by learning context structures of images. To train the CNN we define a set of Jigsaw puzzle permutations and assign an index to each entry. We randomly pick one such permutation, rearrange the nine 75 $\times$ 75 tiles extracted form the input image accordingly, and ask the CNN to return a vector with a probability value for each index. Given nine tiles, there are 9! = 362,880 possible permutations. From [3] we found that the permutation set is an important factor on the performance of the representation that the network learns. The permutation set controls the ambiguity of the task. If the permutations are close to each other, the Jigsaw puzzle task is more challenging and ambiguous. We retrieved the corresponding authors' code, which provided an algorithm to generate the set of permutation and associate them to an index, which was the pseudo label. This way we were sure the task wasn't too ambiguous nor too simple. The number of selected permutation was set to 500, but [3] used 1000, which was computationally unfeasible for us. As it regards the network's structure, we used the same convolutional layers as in the supervised task, with different fully connected section at the end, in order to classify the permutations. Also in this case Adam optimizer was utilized but with learning rate 0.0002, the loss criterion was still Cross Entropy and the number of epochs was 50. The network was trained on the entire train set without any of its labels. The accuracy obtained on the test set was 78%.

(a) Shuffled sample image

(b) Recomposed sample image.

Figure 8. Jigsaw puzzle permutation and its classification result.

The network learnt to solve the puzzle, and so we would expect that its features can be useful to tackle our food classification task. To do so we replaced the fully connected part with a linear classifier able to tackle the 251 classes problem from the supervised learning task. We trained it with the ground truth labels for 20 epochs and an accuracy of 10% was achieved.

5. Comparisons

Even though self supervised learning results are quite worse with respect to the supervised ones, we have to consider that only 500 permutations were used. In fact

when we tried with 100 permutations, the network was perfect in solving the pretext but the extracted features were completely useless in our food classification task. Raising to 200 permutations worsened the pretext solving capability but increased the classification accuracy overcoming the random guess threshold. 500 seemed a good trade-off between computation cost and food classification. Developing furtherly this self supervised approach could allow to increase the training set size substantially also using unlabeled images, keeping the human annotated inputs to reasonable size only for the linear classifier part.

6. Conclusion

We explored food classification using Convolutional Neural Networks (CNNs) in both supervised and self-supervised learning approaches. This study has yielded insights into their capabilities and limitations.
Supervised learning has shown a higher accuracy, but its success is heavily dependent on the quality and quantity of human annotated data. In our case, there was a large number of classes with the majority of samples being outliers, compromising our study. For further researches, a finer labelling of the training and validation data would boost the results.
Self supervised learning, which leverages unlabeled data to learn useful representations, could show promising results, especially when labeled data is scarce. In general self supervised methods exhibit strong generalization capabilities with the final classification part requiring less training examples than the whole supervised CNN. For further researches, instead of more refined labelling, a simpler outlier detection in training and validation set would raise the quality of the training of this method. Moreover, designing more effective pretext tasks suited to our framework, would enhance the effectiveness of our self supervised learning method. For example, at the beginning of our study we thought about using a set number of rotations as a pretext task, but the variability of food images, such as various angles at which the image was captured, let us think that was not suited to this specific data set. However, this variability poses a significant challenge for both supervised and self supervised models and addressing it would require more sophisticated processing techniques and more robust model architectures.

7. Disclosure statement

Benedetta Bensi and Stefano Carotti assure that this project is entirely original without any plagiarism. The research detailed in this report was independently conducted by the authors, with all sources cited in the References Section. No content was AI generated.

References

[1] Kaur, Parneet and and Sikka, Karan and Wang, Weijun and Belongie, serge and Divakaran, Ajay, *FoodX-251: A Dataset for Fine-grained Food Classification*, arXiv preprint arXiv:1907.06167, 2019.

[2] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, 2011

[3] Mehdi Noroozi and Paolo Favaro, *Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles*, ECCV, 2016

[4] Simone Bianco, *Lectures' slides of Supervised Learning*, Università degli Studi di Milano Bicocca, 2024