iFood 2019 Dataset Classification

Annalisa Di Pasquali
858848
Milano-Bicocca
a.dipasquali@campus.unimib.it

Michele Banfi
869294
Milano-Bicocca
m.banfi3397@campus.unimib.it

***Abstract*—The paper explores the classification of a food dataset [1] in a supervised learning context. The automatic identification of food items can assist for applications in dietary monitoring and promoting healthy eating habits. Food classification presents considerable challenges due to the extensive number of food categories, the high visual similarity among different food items, and the scarcity of sufficiently large datasets suitable for training deep learning models. The approach employed in this study addresses these challenges by utilizing a CNN-based architectures to classify images into 251 distinct food categories.**

1. Introduction

Convolutional Neural Networks (CNNs) have proven to be highly effective in image classification tasks, making them a suitable choice for the complex problem of food classification. The initial phase of this project involves the application of a CNN to classify food images into 251 distinct categories. To improve the robustness of the model, the problem is also approached using Self-Supervised Learning (SSL) techniques, to enable the model to learn useful features from data. This approach can potentially enhance the performance of the model by providing a richer representation of the food items.

In section 2 an overview of the dataset used is presented. In the section 3 the preprocessing steps to prepare the dataset for training are discussed. This includes for example normalization and augmentation techniques.

In section 4 the architectures of the Convolutional Neural Networks employed in this study are described in detail. In the section 5 the adaptation of the classification task into a Self-Supervised Learning framework is reported. The results of the project are presented in section 6 including the performances metrics of both the supervised CNN model and the SSL-enhanced model. Comparisons and analyses of the results are provided to highlight the effectiveness of each approach. In section 7 potential future works are discussed. This includes improvements to the models, expansion of the dataset, and exploration of additional machine learning techniques. Finally the section 8 highlights the importance of the results for advancing food classification techniques and enhancing dietary monitoring applications.

2. Dataset

The dataset utilized in this project is composed of instances from 251 distinct food classes. It was created through web crawling, and it included a significant number of incorrectly labeled images, so a careful preprocessing to ensure data quality was needed. The dataset is divided into training and evaluation sets. Due to the absence of a predefined test set, the evaluation data was replaced as the test set. Consequently, a new evaluation set was generated from the training data. Specifically, 25% of the training set was randomly selected without reemission to form the new validation set. To ensure that all 251 classes were represented in the validation set, at least one image per class was included before performing the random selection. Finally, in order to allow the use of the pytorch `ImageFolder` utility, images were organized into directories named after their respective classes. The following tree structure illustrates the organization of the processed dataset:

```
├── ProcessedDataset
│   ├── processed_train_set
│   │   ├── 1
│   │   │   ├── ...
│   │   │   ├── train_059329.jpg
│   │   │   ├── ...
│   │   ├── 2
│   │   ├── ...
│   ├── processed_test_set
│   └── processed_val_se
```

After this process the dataset was divided into three sets with the following size:

- Training: 88977
- Validation: 29600
- Test: 47976

This structured approach ensured a balance representation of classes across the training, validation an test sets in order to obtain an accurate evaluation of the classification model.

The following figure displays how many examples were available for each class:

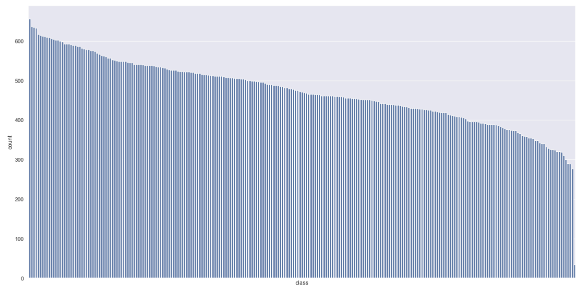


Fig. 1: **Sample per class**: on the y-axis the number of samples per class

The samples for each class were approximately *normally distributed*, with few classes having as many as 600 samples and some having fewer than 100 samples.

Additionally, two examples of training images with very different dimensions are presented below to illustrate the variability presented in the dataset.

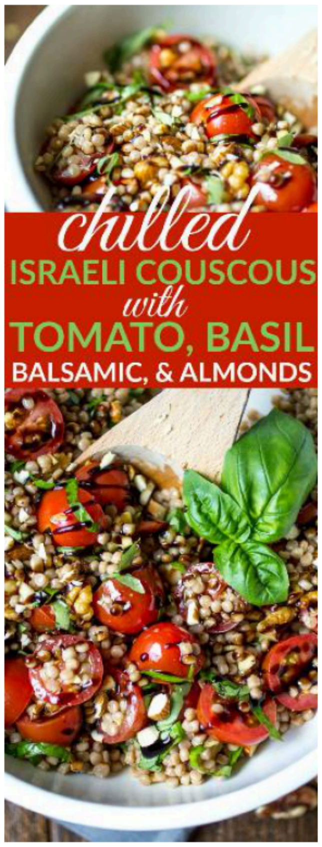


Fig. 2: Train image sample

Fig. 3: Train image sample

To address this imbalance, the following section presents the preprocessing steps performed on the dataset, such as size dimension normalization and data augmentation techniques used to increase the number of samples in the underrepresented classes.

3. Preprocess

To achieve better results and an overall performance of the model, preprocessing of the dataset is essential. As shown in the above section, some images presented very different dimensions. To ensure that the network processed them properly, all images were resized to 128 x 128 pixels, providing a uniform input size.

Additionally, the dataset contains unbalanced classes. To address this issue and enhance the representation of the underrepresented classes, data augmentation techniques were applied. These techniques help increase the diversity and number of samples in these classes, improving the model's ability to generalize.

The following transformations were used:

- Flip: random horizontal or vertical flips.
- Blur: Gaussian blur is applied to slightly smooth the images and reduce noise.
- Rotation: random rotations by certain angles to introduce variability in orientation.

The following is an example of data augmentation applied to the class `marble_cake` is presented:

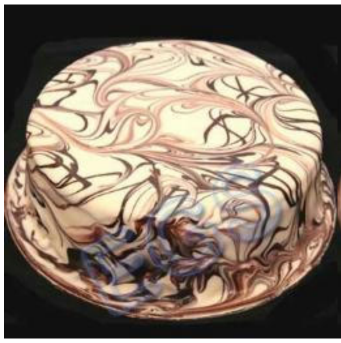


Fig. 4: Original image

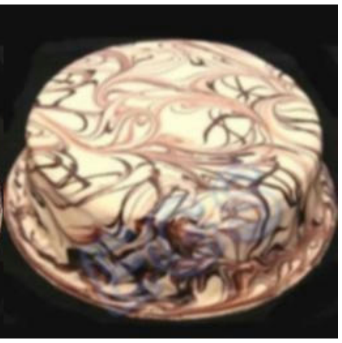


Fig. 5: Blurred version

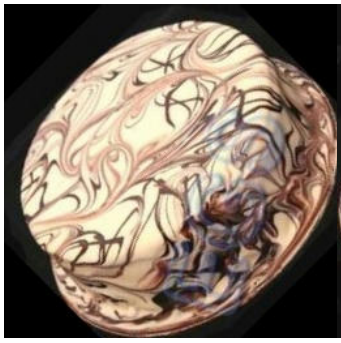


Fig. 6: Rotated version

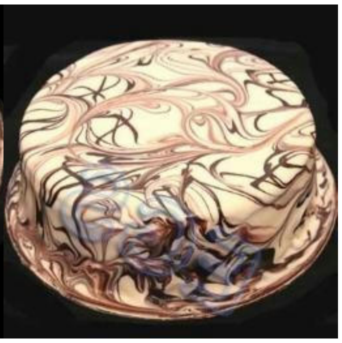


Fig. 7: Flipped version

Furthermore, the mean and standard deviation of the color channels were computed for all training images. These statistical values were crucial for normalizing the images during dataset loading, ensuring that the color distribution remained consistent across all images.

Here are the computed mean and standard deviation values for the dataset:

```
mean = [0.6388, 0.5446, 0.4452]
std = [0.2252, 0.2437, 0.2661]
```

The following images show the effect of normalization on a training image:

Fig. 8: Original image

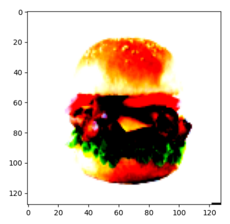


Fig. 9: Normalized image

4. CNN

4.1. CNN's Architecture

The initial phase of the project involved developing a Convolutional Neural Network to classify the images into 251 classes.

The only constraint imposed was the number of parameters: the architecture was limited to have no more than one million parameters. To respect this requirement a deeper Convolutional Neural Network was designed in order to use fewer parameters. The chosen architecture of the CNN consisted of multiple layers, including convolutional layers, pooling layers and fully connected layers. These types of architectures are inspired by GoogLeNet [2].

Several architectures were tried in this study, according to the results obtained, just two of them were chosen to be compared.

In general, the convolutional layers detected the local patterns and the important features in the images, using filters which extract hierarchical features at different levels of abstraction. Then pooling layers were used to reduce the spatial dimensions, in order to decrease the computational complexity and reduce the possibility of overfitting. In the end, fully connected layers were applied for combining the features extracted and perform the final classification.

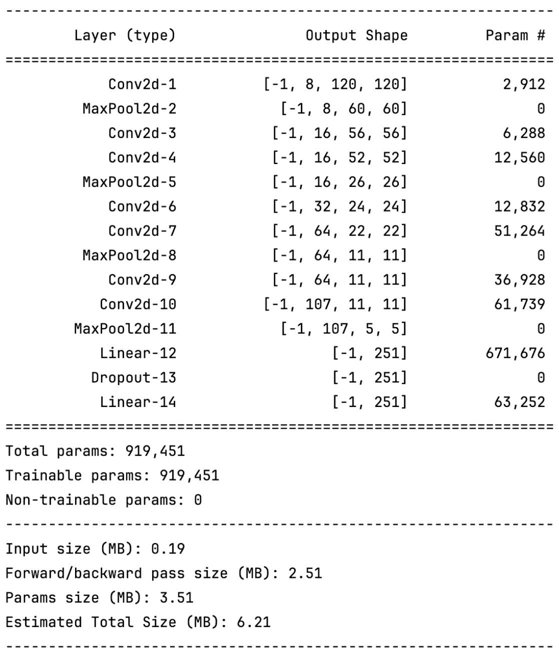
The two selected CNNs differ mainly for the number of parameters, one resulting smaller compared to the other one, and for that reason, one has been referred as *Big CNN* and the other one as *Small CNN*.

4.1.1. Big CNN

The activations functions used in the convolutional and fully connected layers were the Leaky ReLu (Leaky Rectified Linear Unit), so that non-linearity was introduced in the network; the dropout is applied to the fully connected layer to prevent overfitting.

The Big CNN was trained using the categorical cross-entropy loss function and the Adam optimizer with a step learning rate scheduler.

In particular this architecture of the network consisted of 919.451 parameters. The structure can be visualized using the `torchsummary` package.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 8, 120, 120]           2,912
         MaxPool2d-2            [-1, 8, 60, 60]               0
            Conv2d-3           [-1, 16, 56, 56]           6,288
            Conv2d-4           [-1, 16, 52, 52]          12,560
         MaxPool2d-5           [-1, 16, 26, 26]               0
            Conv2d-6           [-1, 32, 24, 24]          12,832
            Conv2d-7           [-1, 64, 22, 22]          51,264
         MaxPool2d-8           [-1, 64, 11, 11]               0
            Conv2d-9           [-1, 64, 11, 11]          36,928
           Conv2d-10          [-1, 107, 11, 11]          61,739
        MaxPool2d-11            [-1, 107, 5, 5]               0
           Linear-12                  [-1, 251]         671,676
          Dropout-13                  [-1, 251]               0
           Linear-14                  [-1, 251]          63,252
================================================================
Total params: 919,451
Trainable params: 919,451
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 2.51
Params size (MB): 3.51
Estimated Total Size (MB): 6.21
----------------------------------------------------------------
```

Fig. 10: **Model parameters**: the torch-summary of the Big CNN

4.1.2. Small CNN

The smaller CNN was constructed with the same general structure of the Big CNN but with some different parameters: the kernels paddings were removed, the sizes of the kernels were changed and even the number of filters applied the in convolutional layers were enlarged.

Also for this network Leaky Relu activation functions and the Adam optimizer were used. Whereas for the optimizer scheduler a Cosine Annelear was implemented. The changes applied affected the architecture in the number of parameter used, resulting in 387.695 parameters.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 8, 120, 120]           1,952
         MaxPool2d-2           [-1, 8, 60, 60]               0
            Conv2d-3          [-1, 16, 54, 54]           6,288
            Conv2d-4          [-1, 16, 48, 48]          12,560
         MaxPool2d-5          [-1, 16, 24, 24]               0
            Conv2d-6          [-1, 32, 20, 20]          12,832
            Conv2d-7          [-1, 64, 16, 16]          51,264
         MaxPool2d-8            [-1, 64, 8, 8]               0
            Conv2d-9            [-1, 64, 6, 6]          36,928
           Conv2d-10           [-1, 128, 4, 4]          73,856
        MaxPool2d-11           [-1, 128, 2, 2]               0
           Linear-12                  [-1, 251]         128,763
          Dropout-13                  [-1, 251]               0
           Linear-14                  [-1, 251]          63,252
================================================================
Total params: 387,695
Trainable params: 387,695
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 2.10
Params size (MB): 1.48
Estimated Total Size (MB): 3.77
----------------------------------------------------------------
```

Fig. 11: **Model parameters**: the torch-summary of the Small CNN

4.2. CNNs's results

The networks performances are shown in the following figures:

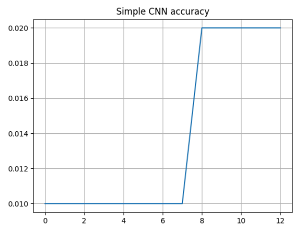


Fig. 12: Accuracy of the Big CNN

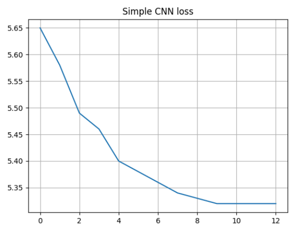


Fig. 13: Loss of the Big CNN

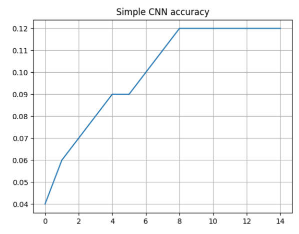


Fig. 14: Accuracy of the Small CNN

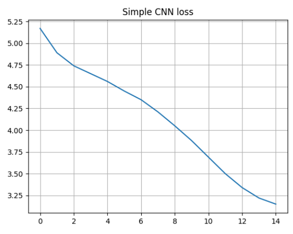


Fig. 15: Loss of the Small CNN

For both of the used architectures the loss decreased but in different manners, notice that the Small CNN's loss decreased more rapidly than the Big CNN's loss. For what concerns the accuracy the Small CNN achieved a better result whereas the Big CNN poorly performed as can be noted from the accuracy plotted.

These behaviors and the results obtained could be attributed to an improperly configured learning rate scheduler: an incorrect scheduler configuration can lead the models to struggle with convergence and stabilization.

5. Self-Supervised Learning

To cast the problem as Self-Supervised Learning (SSL), the network was trained on another simpler task.

The dataset was modified and each image was divided in 4 pieces. For each image, the pieces were shuffled and the order of the shuffled pieces was used as label. The task was the classification of the permutation applied to an image.

Since the images were divided into 4 pieces, the possible permutations were 24. Here an example of an image and the shuffled version.

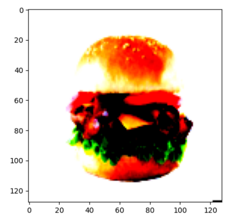


Fig. 16: Original

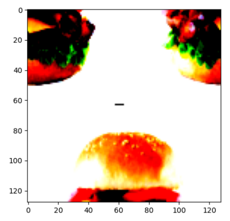


Fig. 17: Shuffled

The permutation of above shuffled image example is `(3, 2, 0, 1)`.

The networks performances on the new task are reported in the following

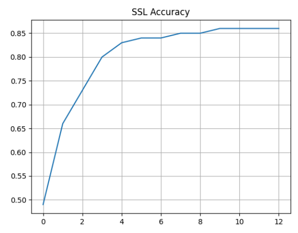


Fig. 18: SSL Accuracy of the Big CNN

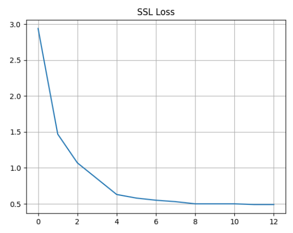


Fig. 19: SSL Loss of the Big CNN

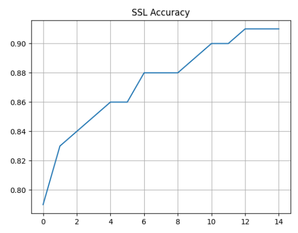


Fig. 20: SSL Accuracy of the Small CNN

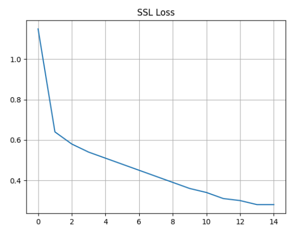


Fig. 21: SLL Loss of the Small CNN

From these plots it's clear that both networks learned the new task assigned properly combining an exponential decreasing losses with a well exponential increase in accuracies.

After the successful results obtained in the SSL task, the weights of the networks were then used as starting points to train again the CNNs on the same training set; since two different CNNs were used, two sets of weights were transferred to the respective CNN. To perform this operation the output of the last layer of the network was changed and from 24 was set to 251.

The results of the original CNNs networks with random initialized weights and the new CNNs networks with SSL initialized weights are shown in the following section.

6. Results

In this section are compared the results obtained from the CNNs initialized with random weights and the CNNs initialized with the SSL-calculated weights.

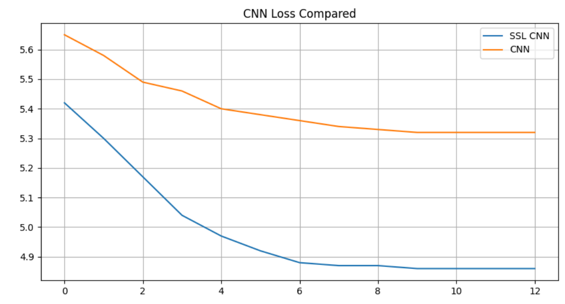


Fig. 22: Losses compared of the Big CNN

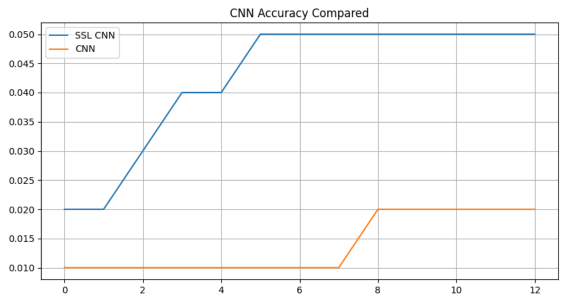


Fig. 23: Accuracies compared of the Big CNN

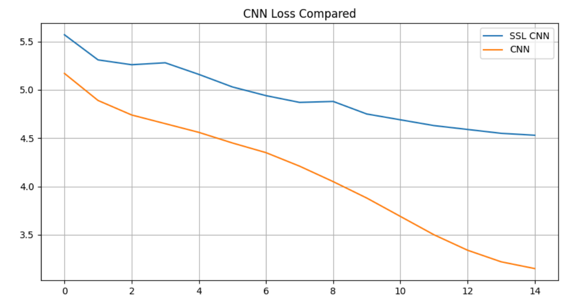


Fig. 24: Accuracies compared of the Small CNN

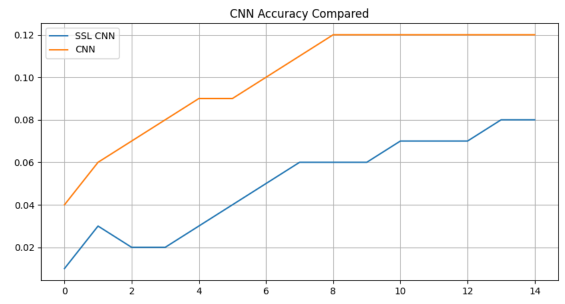


Fig. 25: Accuracies compared of the Small CNN

From the results can be noticed that the Small CNN network underperformed after the use of SSL's weights whereas the Big CNN improved in both accuracy and loss. This behavior can be linked to the theory behind the SSL.

Generally, the SSL task should help the network to learn important features for the classification task and for that reason an improvement in the CNN's performance is expected. The results obtained in this study showed that this holds true for the Big CNN, but it is not the case of the Small CNN.

In our opinion this behavior is related to the number of parameters used in the networks. The Small network learned some general features that lead to a higher accuracy for the initial phase, which could be for example the colors of the images. The addition of the weights computed by the SSL task probably forced the network to change the features to learn, trying to find peculiar features that could improve the performance. But these more specific features are not the ones learned in the first task. On the other side, the opposite case is happening for the Big CNN, since it had already learned those kind of features due to the high number of parameters that could be trained, then the use of the SSL initialized weights helped it to converge in that direction faster.

TABLE I: RESULTS' COMPARISON

	F1 score	Accuracy	F1 score SSL	Accuracy SSL
Small CNN	0.113	0.125	0.054	0.077
Big CNN	0.004	0.015	0.030	0.049

As showed before in the plots, also the F1 score and the accuracy on the test set for both networks remarked the behavior presented above: the Small network learned some features that were not the same as the ones that SSL exploited for the improvement of the CNNs. Probably the SSL took the Small CNN to a local optima from which it was not be able to exit.

To get more insight in the performance of the models, are presented some additional results for the two networks.

The confusion matrix after the first task for the Small CNN is reported. On the *x*-axis the predicted classes are present, while the true labels are on the *y*-axis. Despite of the large number of classes in the dataset, it is notable that on the diagonal an higher number of matches are present.

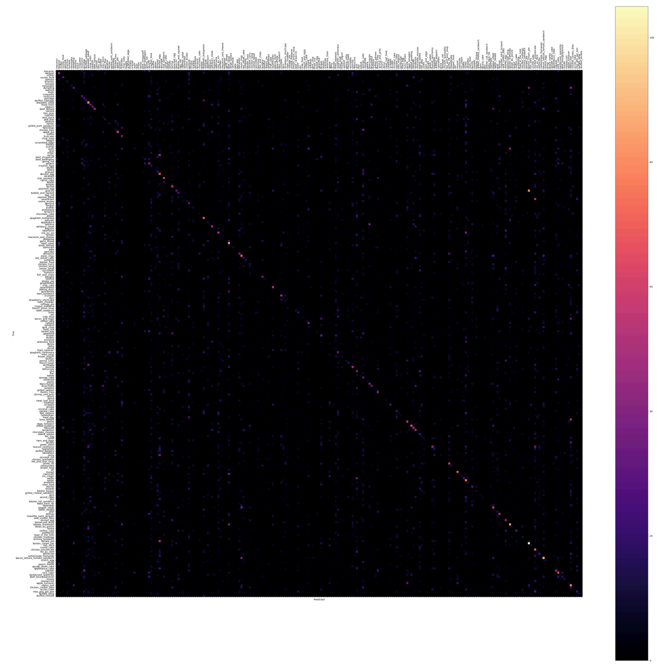


Fig. 26: Losses compared of the Big CNN

Furthermore the top 10 misclassified classes on the test set were computed with the number of misclassification. Here are shown the ones of the Big CNN before and after the SSL initialization.

Before the SSL weights initialization:

1) chicken_provencale: 4626
2) oyster: 3797
3) bacon_lettuce_tomato_sandwich: 2659
4) seaweed_salad: 2169
5) taco: 2153
6) sloppy_joe: 2033
7) fruitcake: 1934
8) fried_calamari: 1879
9) pizza: 1828
10) tenderloin: 1818

After the SSL weights initialization:

1) cruller: 326
2) cupcake: 311
3) knish: 303
4) samosa: 297
5) porridge: 287
6) dumpling: 279
7) seaweed_salad: 273
8) cannoli: 264
9) cockle_food: 257
10) couscous: 256

From what can be seen in the lists, before the SSL task the misclassified classes have a misclassification count more spread apart, ranging from 4626 to 1818, were after was ranging from 326 to 256. Also can be seen that the misclassified class are different except for the *seaweed_salad*. Supporting the hypothesis that the SSL task helped the Big CNN.

7. Conclusions

The results obtained in this study are not really promising due to some motivations: the dataset is contaminated and the restriction of one million parameters limited the dimension of the networks.

With further inspection of the dataset we discovered that there are a lot of contaminated samples in the dataset, reaching some extreme cases where the correct images are fewer than the contaminated images, for example the class `poi`.

For solving this task we tried several configurations of the networks and different parameters, for example for scheduler, optimizer, batch size, number of epochs and so on. The results obtained with those two architectures are not the best result in accuracy and F1 score in an absolute way, but were chosen to showcase the differences in the results that a network architecture can cause.

Furthermore higher achieving architectures on the initial CNN task were found but could not be well compared since they did not perform well on the SSL task; so *"worse"* networks were used for both the initial task and the SSL task to allow a fair comparison.

8. Future works

The overall performances of the methods used can be further improved with some shrewdness.

It needs to be said that the restriction imposed on the network parameters posed a challenging problem to the construction of a decent network. Further fine tuning is needed to enhance the performance of the network, specifically to the optimizer and the scheduler, since from the runs performed those values were responsible for the goods results of the networks. As stated before enlarge the networks and clean the dataset can lead to overall better performing in both the CNNs presented.

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] C. Szegedy *et al.*, "Going Deeper with Convolutions." 2014.