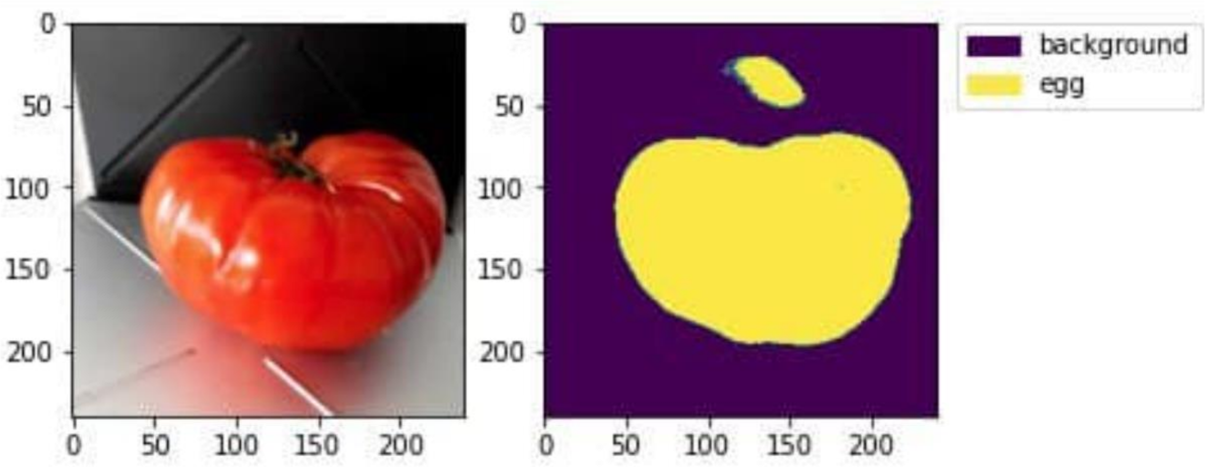
Report of project
"Food recognition"



"maybe not so much"

Prof. **Andrea Asperti**
A.Y. **2020/2021**

Name	Surname	Matricola	Academic mail
Pavlo	Seroyizhko	982598	pavlo.seroyizhko@studio.unibo.it
Xiaowei	Wen	982501	xiaowei.wen@studio.unibo.it

GitHub repository: https://github.com/WenXiaowei/food-recognition-project

Introduction

The Food Recognition Challenge has always been a hard task for data scientists. First for all, for the enormous number of types of food that exists. Secondly, for extreme similarity of certain classes, that even humans aren’t able to distinguish.

Semantic image segmentation is a hard challenge and Deep Convolutional Neural Networks are complex models with millions of parameters to estimate, therefore they require huge datasets and fine tuning, which are both difficult, time consuming tasks and they require huge computational power.

Lately there were proposed several models that showed great results in image segmentation such as FCN, U-Net, DeepLab etc. We have chosen The One Hundred Layers Tiramisu model to see how it would perform in this challenge.

Data generation

Due to the enormous size of the dataset we have decided to use the python generator function. Such functions on next() function calling generate the number of inputs and outputs equal to the desired batch size. To manipulate the COCO annotations we used the Official APIs for the MS-COCO dataset pycocotools 2.0.2. As was suggested, we chose the first 16 most frequent classes, excluding “coffee-with-caffeine" and "espresso-with-caffeine" due to their similarity.

One of the challenges was the organization of the classes. Since the categories were neither sequential nor ordered, we came up with a simple data structure, shown below, which sums up the categories that we included in the training set, their IDs in COCO annotation, and class id used for output.

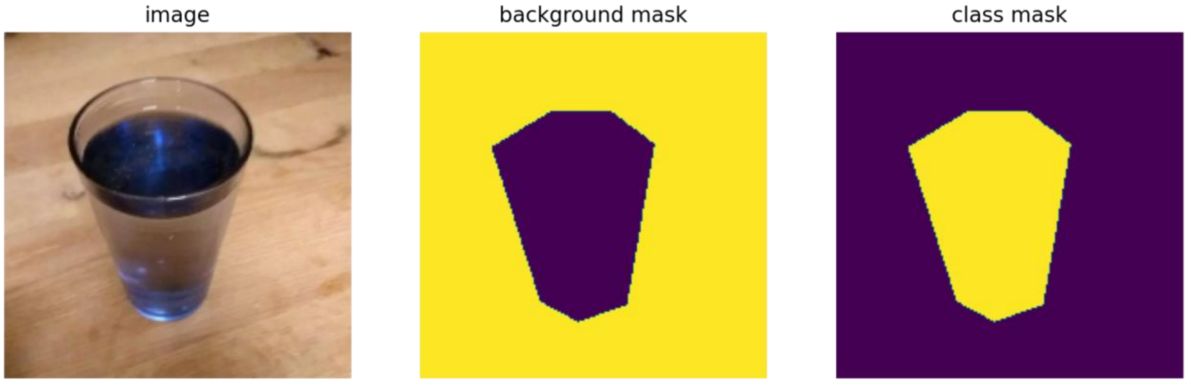
Category name	**Category ID**	**Class ID**	**Number of images**
background	0	0	0
water	2578	1	1837
bread-white	1566	2	1276
salad-leaf-salad-green	1040	3	1190
tomato	1069	4	1072

butter	2053	5	1010
bread-wholemeal	1565	6	905
carrot	1078	7	898
rice	1468	8	659
egg	2022	9	632
mixed-vegetables	1022	10	627
wine-red	2618	11	546
jam	2099	12	505
apple	1151	13	504
potatoes-steamed	1010	14	452
banana	1154	15	412
cheese	1311	16	405

The input for CNN is a set of RGB images whose values are normalized. The output is a set of the same size of the binary masks with 17 channels. To sum up, shapes are:

$$X.shape \ = \ [batch\ size, 192, 192, 3]$$
$$y.shape \ = \ [batch\ size, 192, 192{,}17]$$

The background of the output (channel 0) is formed as all 1s where the category mask is absent. The example of how the input and output were formed is represented in the image below.

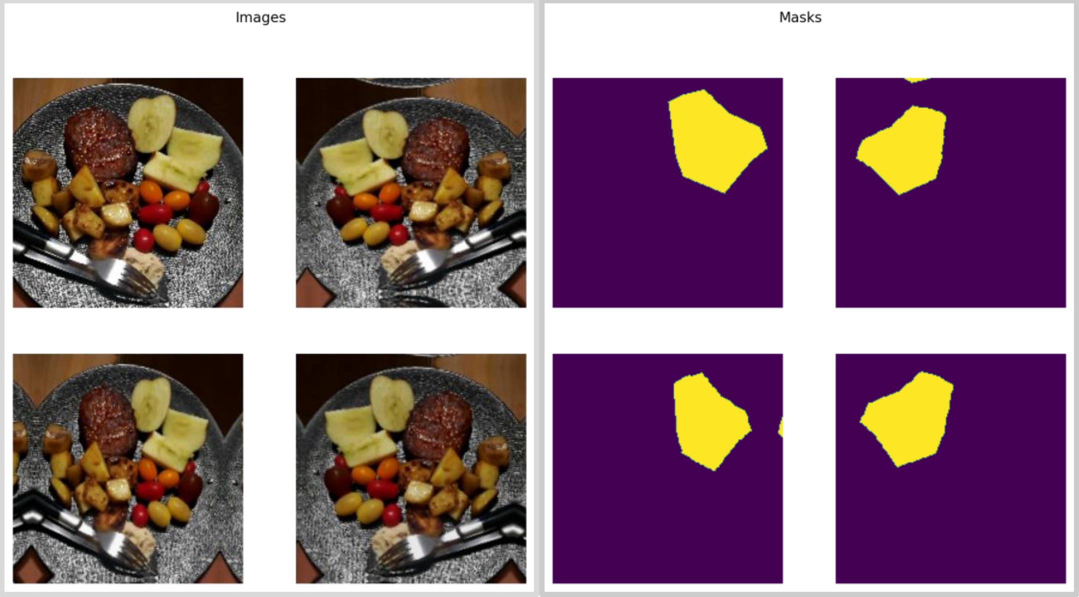


Augmentation

For experimental purposes we also tried training with augmented images. The augmentation features include:

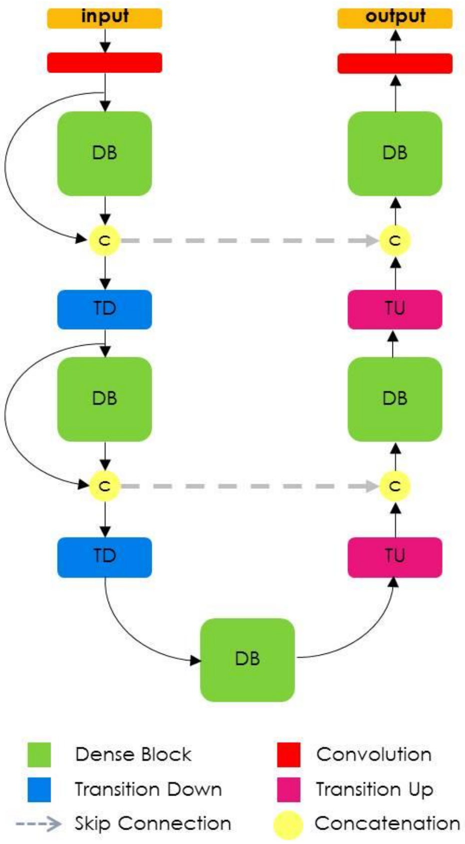
- rotation in range of 5°
- horizontal shifting of 1%
- vertical shifting of 1%
- brightness in range (0.8,1.2) (not applied to the masks)
- shear in range of 1%
- zoom in range (1, 1.25)
- horizontal flip
- vertical flip

On each batch one of the images remains original and others are modified versions of it with randomly applied augmentation. Here is an example of a batch of size 4.

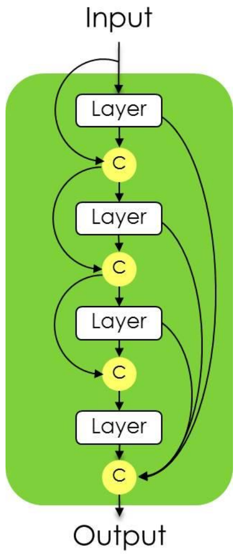


Model

As the model we chose a CNN called The One Hundred Layers Tiramisu (Jegou et al., 2017). The model has shown to be very efficient in image segmentation and have reached the state-of-art on urban scene benchmark datasets such as CamVid and Gatech, without any further post-processing module nor pretraining. Also, we could not avoid the irony of using a model called Tiramisu in the Food Recognition Challenge.
The model has architecture as the image on the left.



Dense Block (DB) is implemented as follows: A first layer is applied to the input to create k feature maps, which are concatenated to the input. A second layer is then applied to create another k features maps, which are again concatenated to the previous feature maps. The operation is repeated 4 times. The output of the block is the concatenation of the outputs of the 4 layers, and thus contains 4 ∗ k feature maps.



Where Layer is build as

Layer
Batch Normalization
ReLU
3 × 3 Convolution
Dropout p = 0.2

While Transition Down (TD) and Transition Up (TU) have the following architecture:

Transition Down
Batch Normalization
ReLU
1 × 1 Convolution
Dropout p = 0.2
2 × 2 Max Pooling

Transition Up
3 × 3 Transposed Convolution stride = 2

Experiments

During the training phase, as first thing we tried hyper-parameters suggested by the paper, that is:

- optimizer: RMSProp with learning rate = 0.001.
- loss function: categorical_crossentropy
- metrics: accuracy

with these hyperparameters, the value of the loss function diverges after some training steps and validation accuracy is about 0.20.

Then, we tried using other loss functions:

- “sparse_categorical_crossentropy” which, as the previous one, diverges;
- “bce_jaccard” converges to 0.5, but the model after several attempts of training, still doesn't give any good predicted mask, this loss functions is implemented in Segmentation-models Github repository (Segmentation models);
- “categorical_focal_loss”, as suggested by the paper (Jadon, 2020), guarantees the best performance than the others. Since the beginning the loss value is close to 1, and it converges to 0 when during the last epochs of training.

By using the focal loss, we tried also with different metrics such as IoU_score from segmentation models and the Mean IoU from Keras library, with different result, as the first one computes the IoU score based over the whole batch, while the second by is computing the Intersection over union accumulated since the beginning of the training.

Then, as the previous configuration produces good results in terms of class prediction and the segmentation meaning that the down-sampling is working as well as the up-sampling path. So, the model has been trained with different number of epochs:

- 50 epochs of one thousand images per each **without** image augmentation.
- 50 epochs of one thousand images per each **with** image augmentation
- 75 epochs of one thousand images per each **without** image augmentation.
- 75 epochs of one thousand images per each **with** image augmentation.
- 100 epochs of one thousand images per each **without** image augmentation.

Results Comparison

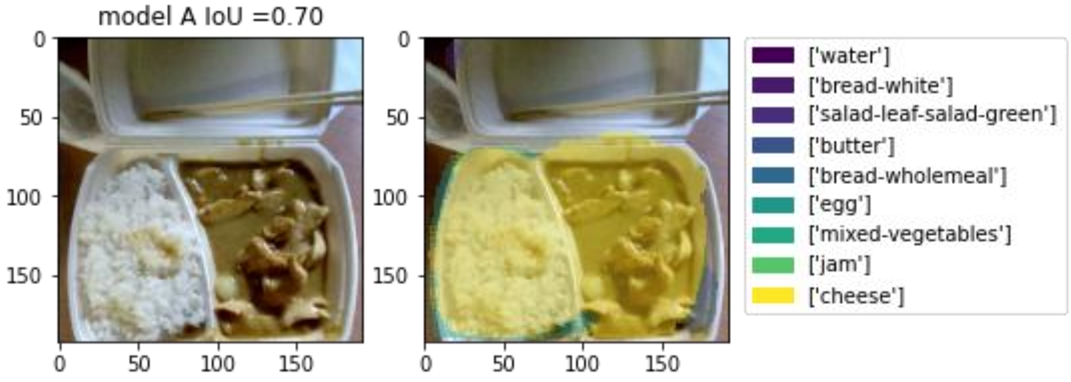
Comparison of the model with optimizer RMSprop (learning_rate = 0.001), as loss focal loss, as metrics mean IoU from keras library, the same model was trained as follows:

Alias	N. Epochs	Image Augmentation	Mean IoU score *	Precision	Recall	F1 Score
A	50	No	0.73	0.54	0.94	0.68
B	50	Yes	0.75	0.57	0.93	0.70
C	75	No	0.70	0.50	0.89	0.64
D	75	Yes	0.75	0.62	0.80	0.70
E	100	No	0.71	0.54	0.78	0.64

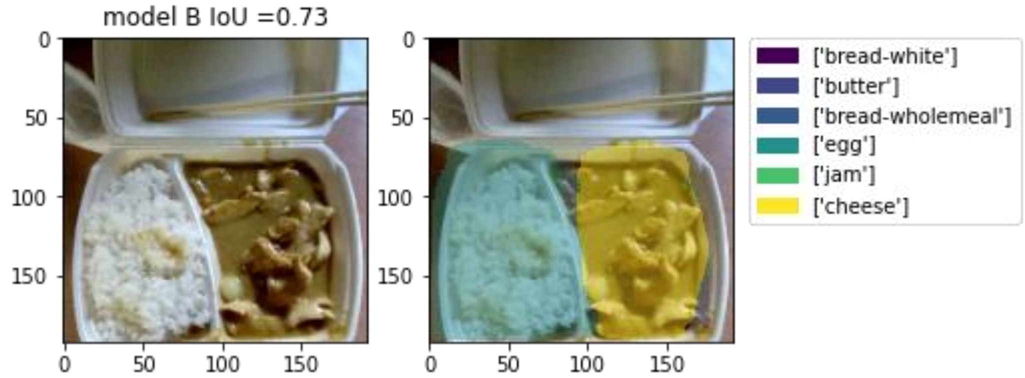
(* Scores obtained testing the model over 458 images from the test dataset)

Then, as we observed a lack of training with 50 epochs models, and an overfitting problem with model E, so we tried with 75 epochs, once with augmentation and not. Earning an improvement of the IoU score of 0.05 with the model D.

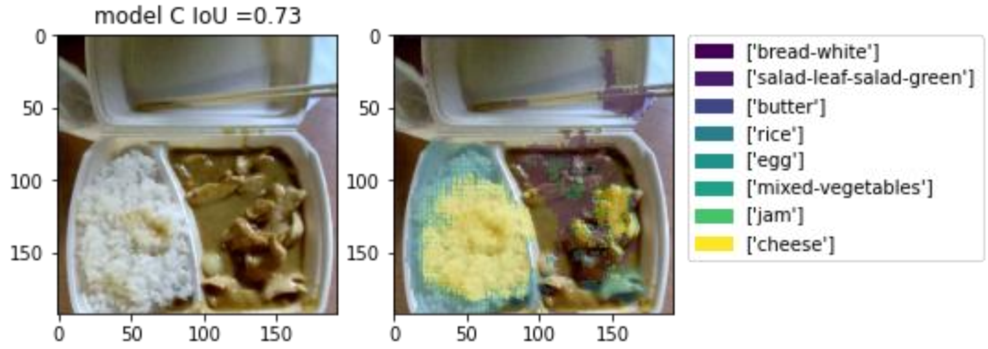
We can notice that the model with E has the same score of the one with the model B. Now, we are going do some comparisons of predicted masks:
As we can see, the models perform quite well in some situations and not good in others, it depends on many factors.



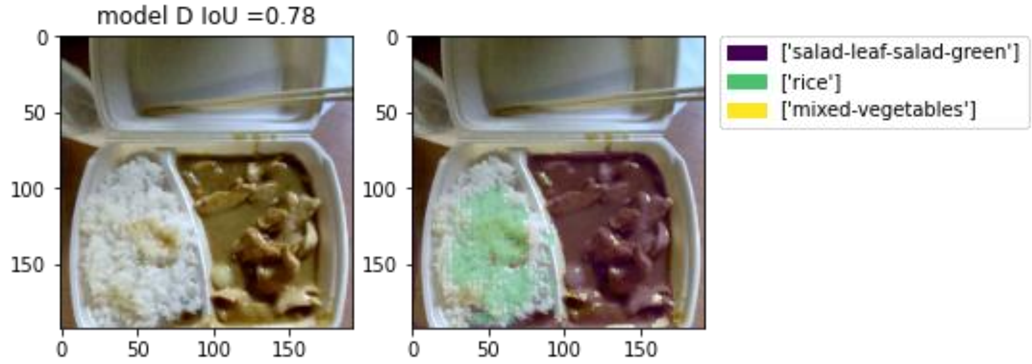
Model A performs the segmentation task quite well, at least differentiating food from not food, but the classification task is not performed perfectly as it is predicting wrong classes.



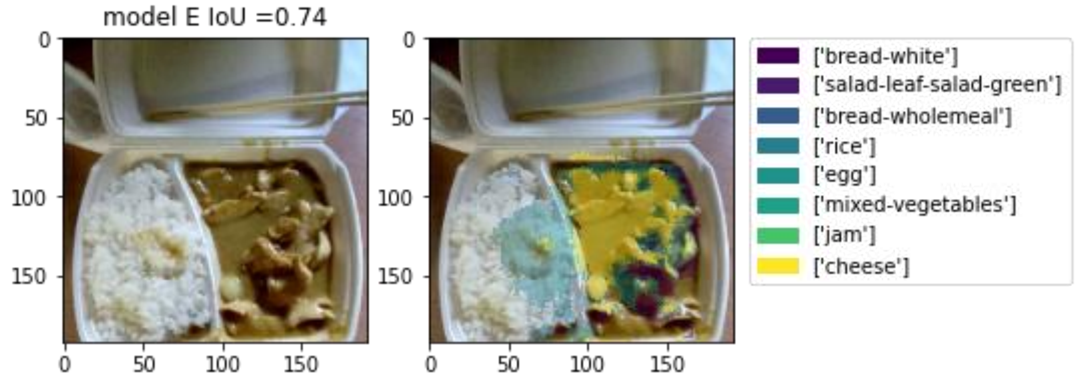
Model B also performs the segmentation task quite well, also separating the two main parts of the food, but the classification task is not performed perfectly as it is recognizing rice as egg.



Model C is performing the segmentation task better than the previous two, because it's recognizing that in the major part the right part of the food is not labelled, and the in the left part, the classification task is done quite well because it's classifying a portion of a rice correctly.



Model D is the one that performs better than the others, as it segmented the image into two main parts, left one and right one, and classifying the left one correctly. And although the right part is wrongly classified but it is segmented almost perfectly from the container of the food.

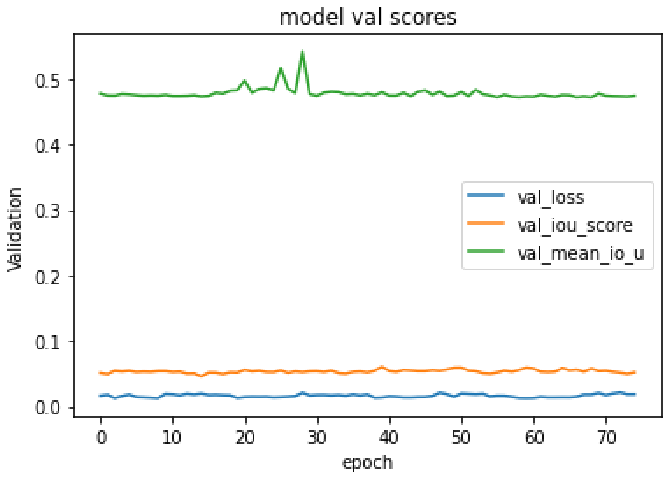


The model E's segmentation ability can be considered ok, as it's segmentation evidences the major part of the food, and also it predicts correctly the class rice, but on the right part, it's performing roughly.

Best model: model D

After the comparison, based on the metrics, we decided that the best model is the model D, for the following reasons:

- Its segmentation ability is good, it can recognize almost all the food.
- Its prediction ability is also good, because it's not predicting too much false positive, as the consequence, the classification give as result only few (correct) classes



The difference between the data obtained during the training-validation and the one given in the table is caused by the fact that we also used a threshold to binarize the probability of a pixel to be in a certain class. In this way, on average, we earn about 0.2 of IoU score.

We can notice that the validation mean-IoU has a peak at a certain point, precisely at epoch 28, so we performed another test, with only 28 epochs of training keeping other hyper-parameters the same and it does not yield any precision or mean IoU.

Conclusion

The Food Recognition Challenge is a very interesting exercise both from studying and applying point of views. Sometimes it's quite challenging for humans, so no surprise the machine has a hard time to succeed in this competition. However, the chosen model "The One Hundred Layers Tiramisu", in our opinion, has done well enough. It performed very well in the segmentation task, and worse in the classification one. Most errors were made in topologically close classes such as "salad-leaf-salad-green" and "mixed-vegetables".

To improve the results, there are several things that can be done. First of all, the quality of the annotations of the dataset should be improved. There are a lot of masks in the training set that are both imprecise and incorrect. Secondly, the number of the images per class should be increased. CNNs are complex models, in our case there were nearly 9 million parameters to estimate, therefore 6 hundred images per class, even with augmentation, is not enough to have good results. Finally, we were limited by computational power. Again, for reasons mentioned above CNNs need a lot of time to be trained. Since there is no exact formula for what could work, we had to try several combinations of loss functions, hyperparameters and number of epochs. Therefore, we think that by applying improving techniques mentioned above with enough time and/or with a more powerful computer could make the model functional even in real life applications.

Recognizing food was always a challenge for robots. We can easily see how this can be helpful to people with allergies and intolerances, so we hope someday machines can classify food at least as good as people.

Bibliography

Jegou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. http://dx.doi.org/10.1109/cvprw.2017.156

Jadon, S. (2020). A survey of loss functions for semantic segmentation. *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. http://dx.doi.org/10.1109/cibcb48159.2020.9277638