DEEP LEARNING

FOOD RECOGNITION CHALLENGE

September 2021

Francesco Vannoni
Mirko Del Moro

Contents

Abstract

Food recognition is an interesting and active topic nowadays. It could be useful for people to use their mobile, taking photos of what they consume, in order to monitor their food intake. Food tracking can be not only of personal interest but it can also have a medical relevance. The use of such a program requires the ability of it to recognize and segment food from images and eventually also to estimate the dimensions of it.
One of the hardest task is that we should be able to recognize and segment from photos taken by common people and not from professional photographers. Accordingly to it food may be overlapping and mixed up, photos may be not perfectly on focus or they may show only a part of the dish.
Our approach is to use convolutional neural networks to segment images.

1 Introduction

This work deals with food image segmentation, the scope is to segment and recognize all the types of food within the dish photo. In order to do it we use a specific convolutional neural network called U-Net, implemented through the python library Keras.
U-Net was originally invented and first used for biomedical image segmentation. Its architecture can be broadly thought of as an encoder network followed by a decoder network [1]. One of its peculiarity is that it works well also with a relatively small dataset.
With the aim of getting the best results we try several combinations of hyperparameters for the net. In every experiment we are going to change the following parameters:

- number of layers of the u-net architecture
- loss functions and optimizers

The dataset we use is split in two parts: train and validation. Each of them consist of images and relative annotations containing segmentation coordinates and images details. The annotations contains some mistakes because collected by hand from human but also by automatic algorithms. A quite frequent case is annotations presenting rotated segmentation coordinates with respect to the image. These problems are fixed in the preprocessing part. After that we prepare our dataset in order to make it trainable by the model.
In particular we generate masks from the annotations considering also the category of food they belong to (as explained in the next section).
In the end we train the model and evaluate it with Intersection Over Union (IoU) metric computing overlap between the true and the predicted masks for each image.
In computing the previous tasks we follow some approaches due to the limited amount of resources available to us. In particular we use Google Colab Pro providing a 12GB NVIDIA Tesla K80 GPU and 12GB RAM.

2 Preprocessing

Before than using the data for the food recognition challenge some manipulations on it are necessary. In particular we have to prepare the data in order to let it be exploitable by the algorithms used. In addition we found that the dataset contains some mistakes and mismatching dealings between images and corresponding annotations.

2.1 Dataset and Categories

The dataset, named "AICrowd Food Recognition Challenge Dataset", [2] consists of 273 classes of food along with their respective bounding boxes and semantic segmentation masks. It is divided into 2 parts: training and test:

- The training set consists of 24119 RGB images, with their corresponding 39328 annotations in MS-COCO format.
- The test set consists of 1269 RGB images, with their corresponding 2053 annotations in MS-COCO format.

2.2 Dataset Cleaning

Analyzing the dataset we found some problems that may decrease the performance of the model which we are going to train. To deal with it we act as explained below.

2.2.1 Rotated Annotations

MS-COCO format annotations often presents a common problem: the polygon segmentation coordinates are rotated with respect to the relative image (*Figure 1*).

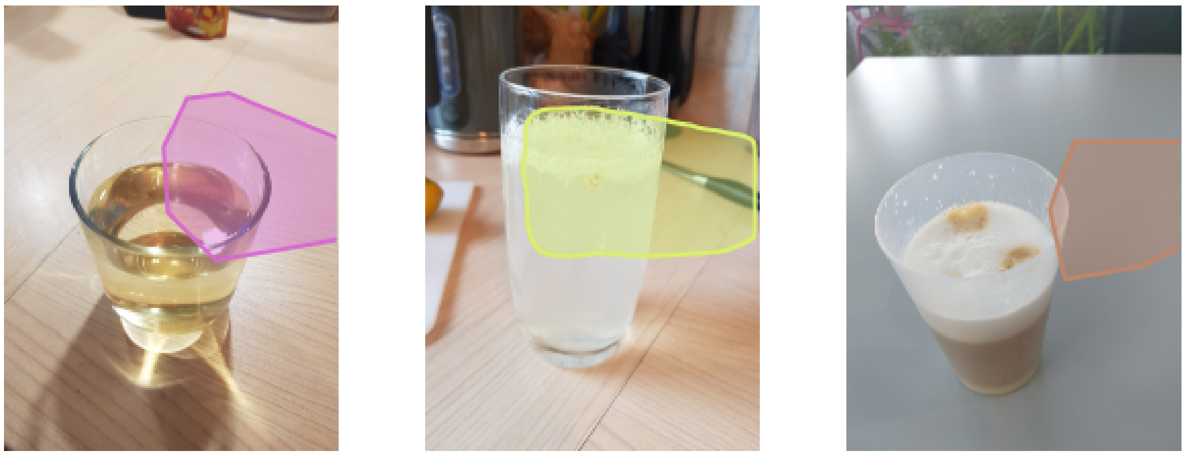

Figure 1: Examples of rotated masks with respect to images

Counting how many these images are, we found out that they are 27 for the training set and 1 for the test set. Since it is a very small part of the entire dataset we decide to remove them.

2.2.2 Uncomplete or Empty Annotations

We check if there are annotations with empty segmentation coordinates and found that every annotations contains at least one coordinates couple.
Therefore we look over all couples of the segmentation coordinates checking if one of them contains only one of the two coordinates. Also in this case there isn't any couple presenting this problem.

2.3 Most Common Categories

Due to the limited amount of resources available to us we decide to use a subset of the 273 categories of the dataset.

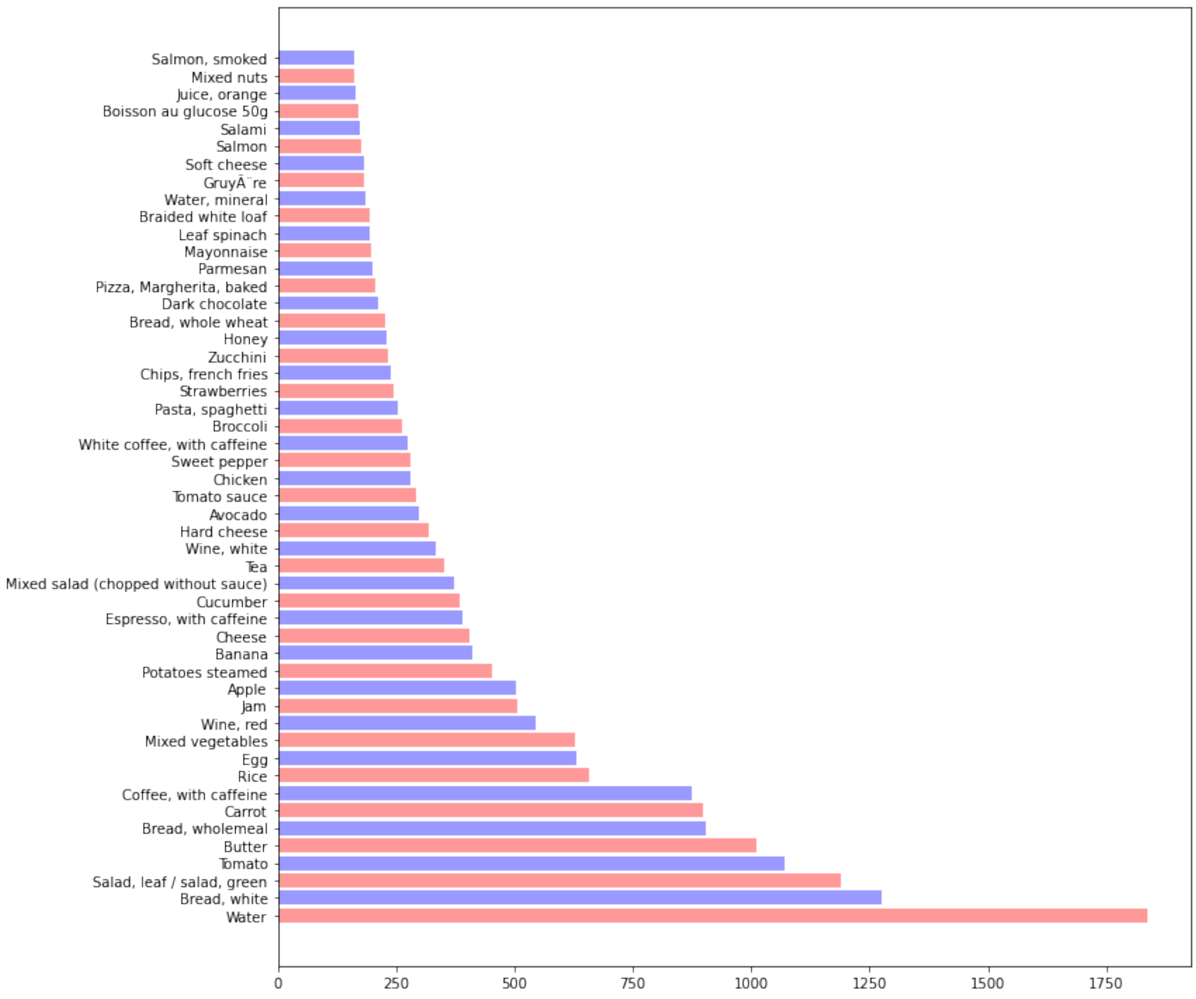

Figure 2: Frequency of most common 50 categories

Inspecting *Figure 2* we observe that the distribution of the annotations among the categories has an exponential behaviour. We want to select a subset having as cardinality a power of 2. We decide to use 16 as number of categories, so we pick the 14 most common categories and we use one category for the background and one for all the others categories. These are the categories with a higher variance of their frequencies, so it represents quite well the dataset distribution.

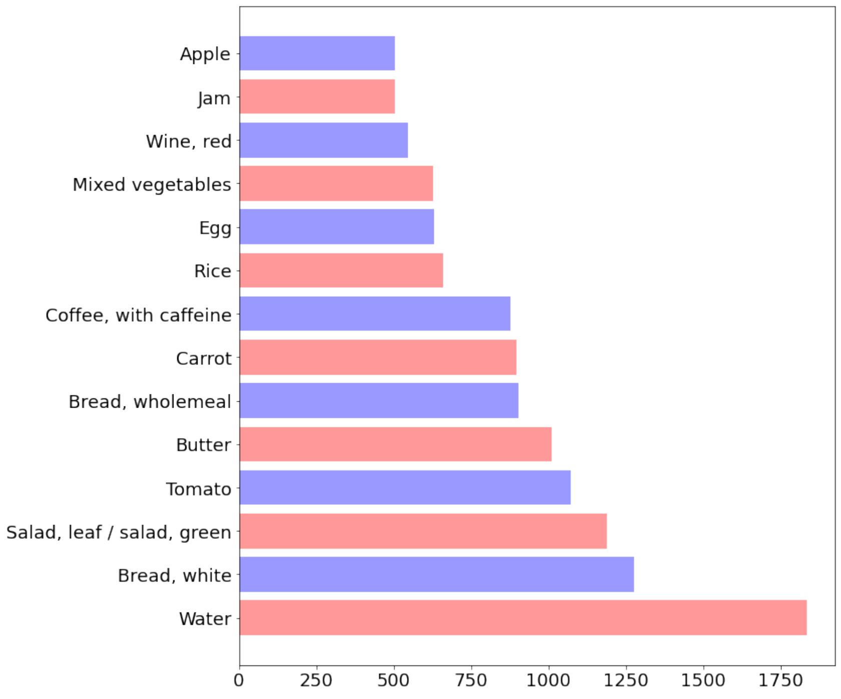

Figure 3: Frequency of most common 14 categories

2.4 Multi-class Masks

Image segmentation is performed by computing masks. With mask we mean a matrix of zero and non-zero values. When a mask is applied to another image of the same size, all pixels which are 0 in the mask are set to 0 in the output image. All others remain unchanged. Indeed, before training we have to transform mask from polygon coordinates (provided in MS-COCO format) to binary mask.
Therefore there are images containing more than one annotations corresponding to different food categories. In addition to that masks corresponding to the same image may also be overlapped between them.
We have to keep separated masks belonging to different categories and so we decide to follow the one hot encoding approach [3] (*Figure 4*).

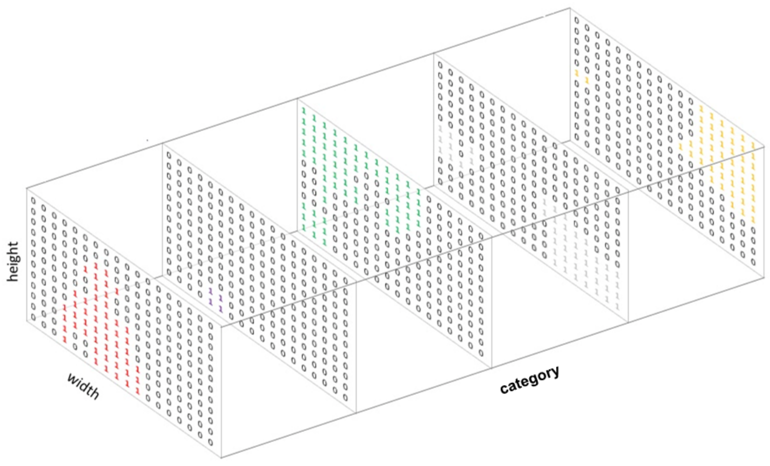

Figure 4: One Hot Encoding

In One Hot Encoding each mask is encoded as a matrix of dimension:

$$\text{SIZE} = W \times \text{H} \times \text{C} \tag{1}$$

where W is image width, H is image height and C is the number of categories.
The first level (index 0) represent the background, the second one the most common category ("Water"), the third one the second most common category ("Bread-White") and so on for the first 14 categories. The last layer is used for all the others categories. This approach is expensive from the memory complexity point of view. An alternative solution could be to encode each mask as in *Figure 5.* In this case each pixel of the

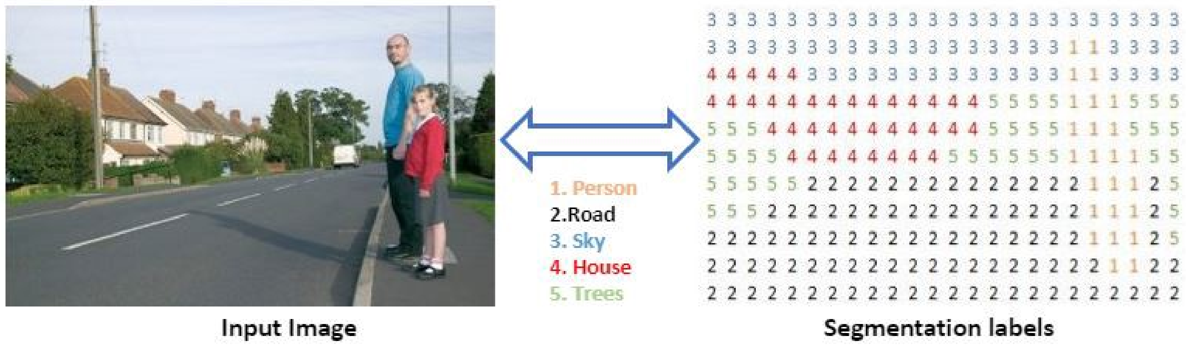

Figure 5: Alternative Mask Encoding

mask is multiplied by an integer corresponding to a category. It is surely a more scalable approach than One Hot Encoding, indeed it doesn't require a matrix having as depth the number of categories to encode every mask. However we tried to use this kind of encoding and we obtained worse segmentation results with respect to One Hot Encoding. The main problem of this approach is overlapping. Many of the training masks indeed overlaps.
A possible techniques to deal with overlapping can be: when a pixels belongs to more than one category we set that pixel to the value corresponding to the most frequent category between the ones overlapping.

2.5 Dataset Generator

"AICrowd Food Recognition Challenge Dataset" [2] is a large dataset containing more than 24,000 images. In order to not store the entire dataset in memory we use generators. Generator functions are a special kind of function that return a lazy iterator. These are objects that you can loop over like a list. However, unlike lists, lazy iterators do not store their contents in memory [4].
Generators require batch size as argument, which is the dimension of the subset that they have to compute. When a generator is called it computes a subset and then statement pauses the function saving all its states and later continues from there on successive calls [5].
After generating the subset we resize all images and the respective masks with a fixed dimension (128 × 128) because our models works on images sharing the same size.

3 Evaluation Criteria

As explained in the "AICrowd Food Recognition Challenge Dataset" [2], for the model evaluation we use Intersection Over Union (IoU) metric.
IoU measures the overall overlap between the true region and the proposed region. Then we consider it a True detection, when there is at least half an overlap, or when IoU $>$ 0.5.
Then we can define the following parameters :

- Precision (IoU > 0.5)
- Recall (IoU > 0.5)

The final scoring parameters AP{IoU > 0.5} and AR{IoU > 0.5} are computed by averaging over all the precision and recall values for all known annotations in the ground truth.

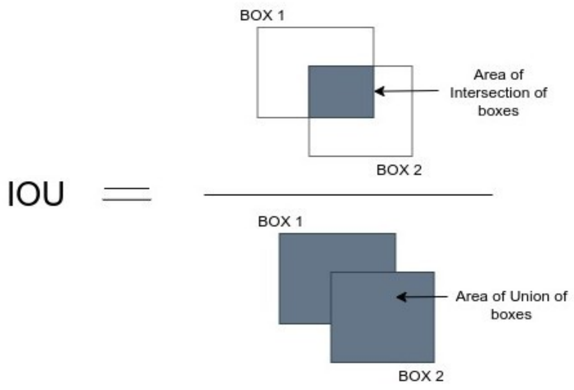

Figure 6: Intersection Over Union

For us intersection over union is a ratio, where the numerator is the area of overlap between the predicted mask and the ground-truth mask and the denominator is the area enclosed by both the predicted and the true mask.

As explicated before we compute precision and recall for each mask where the IoU metric is higher than 0.5. Precision and recall are defined as follows:

$$\text{Precision} = \frac{\text{True positive}}{\text{Total predicted positive}}$$

Where Total predicted positive = True positive + False positive

$$\text{Recall} = \frac{\text{True positive}}{\text{Total actual positive}}$$

Where Total actual positive = True positive + False negative

4 U-Net Model

The model chosen for the task is U-Net. U-Net is a convolutional neural network architecture typically used for image segmentation.
We use the library Keras to implement U-Net model. Keras is an open-source software library that provides a python interface for artificial neural networks. It acts as an interface for the TensorFlow library.

The typical use of convolutional networks is on classification tasks, where the output to an image is a single class label. However, in many visual tasks, especially in biomedical image processing, the desired output should include localization, i.e., a class label is supposed to be assigned to each pixel [6].
The main idea is to supplement a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. Hence, these layers increase the resolution of the output. In order to localize, high resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information. As a consequence, the expansive path is more or less symmetric to the contracting path, and yields a u-shaped architecture The network does not have any fully connected layers and only uses the valid part of each convolution, i.e., the segmentation map only contains the pixels, for which the full context is available in the input image [7]
In U-Net the input and the output share the same size.

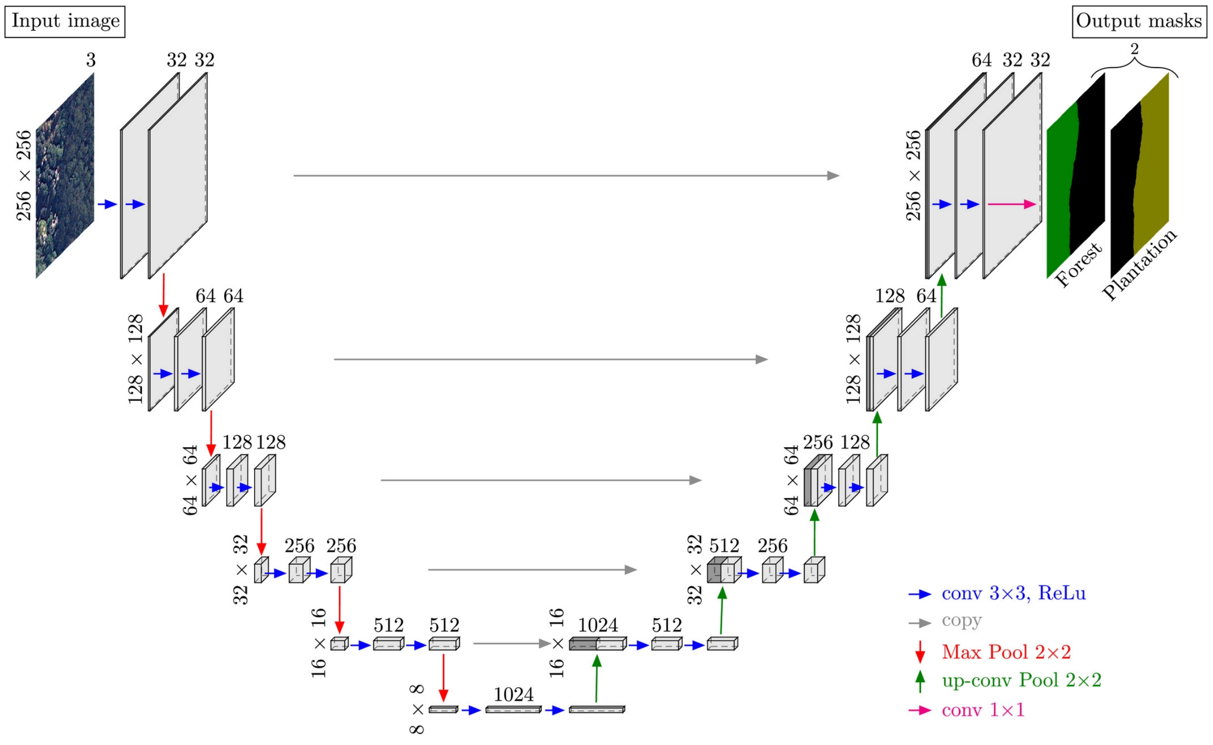

Figure 7: U-Net Architecture

Another challenge in many cell segmentation tasks is the separation of touching objects of the same class. To this end, [7] proposes the use of a weighted loss, where the separating background labels between touching cells obtain a large weight in the loss function. The energy function is computed by a pixel-wise soft-max over the final feature map combined with the cross entropy loss function. We try several combinations of loss functions and optimizers shown in Section 4.5.

4.1 Architecture

The U-net architecture is illustrated in *Figure 7*. This architecture consists of three sections: the contraction, the bottleneck, and the expansion section. The contraction section is made of many contraction blocks. Each block takes an input and applies two 3X3 convolution layers followed by a 2X2 max pooling. The number of kernels or feature maps after each block doubles so that architecture can learn the complex structures effectively. The bottom most layer mediates between the contraction layer and the expansion layer. It uses two 3X3 CNN layers followed by 2X2 up convolution layer. The heart of this architecture lies in the expansion section. Similar to contraction layer, it also consists of several expansion blocks. Each block passes the input to two 3X3 CNN layers followed by a 2X2 upsampling layer. After each block number of feature maps used by convolutional layer get half to maintain symmetry. However, every time the input is also get appended by feature maps of the corresponding contraction layer. This action would ensure that the features that are learned while contracting the image will be used to reconstruct it. The number of expansion blocks is as same as the number of contraction blocks. After that, the resultant mapping passes through another 3X3 CNN layer with the number of feature maps equal to the number of segments desired [8].
In order to bring all the activation values to the same scale, we use Batch Normalization to normalize the activation values such that the hidden representation doesn't vary drastically and also helps us to get improvement in the training speed [6].
The weights are initialized following Gaussian distribution. A padding is added in order to an output shape equal to the input one.

4.2 Activation Functions

An activation function is a function used in neural networks which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold. If the inputs are large enough, the activation function "fires", otherwise it does nothing.
Our U-Net model is implemented with two kinds of activation functions: "Softmax" for the output layer and "ReLU" for all the others [9].

ReLU is a linear function that outputs zero if its input is negative, and directly outputs the input otherwise:

$$f(x) = \max\,(0, x)$$

Softmax is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression. When the result of the network is a probability distribution, e.g. over K different categories, the softmax function is used as activation.

$$softmax(j, x1 + c, ..., xk + c) = \frac{e^{x_j}}{\sum_{j=1}^{k} e^{x_j}}$$

Softmax function is always greater than 0 and lower than 1. In addition to that

$$\sum_{j=1}^{k} softmax(j, x1 + c, ..., xk + c) = 1$$

since we expect probabilites to sum up 1.

4.3 Loss Functions

As shown in Section 4.5 we try three kinds of loss functions looking for the one fitting better our model: "Categorical Crossentropy", "Focal Loss", "Categorical Crossentropy + Dice Loss".

The **Categorical Crossentropy** [10] loss function calculates the loss of an example by computing the following sum:

$$Loss = -\sum_{i}^{n} y_i \times \log \hat{y}_i$$

where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output. The minus sign ensures that the loss gets smaller when the distributions get closer to each other. The categorical crossentropy is well suited to classification tasks, since one example can be considered to belong to a specific category with probability 1, and to other categories with probability 0.

Focal loss down-weights the well-classified examples [11]. This has the net effect of putting more training emphasis on that data that is hard to classify. In a practical setting where we have a data imbalance, our majority class will quickly become well-classified since we have much more data for it. Thus, in order to insure that we also achieve high accuracy on our minority class, we can use the focal loss to give those minority class examples more relative weight during training.
Focal Loss proposes to down-weight easy examples and focus training on hard negatives using a modulating factor as shown below:

$$FL(p_t) = -\alpha_t(1 - p_t)\log p_t$$

The third loss function we use is given by **Categorical Crossentropy + the Dice Loss**.
The **Dice loss** is widely used metric in computer vision community to calculate the similarity between two images [11]. Later in 2016, it has also been adapted as loss function known as Dice Loss.

$$DL(y, \hat{p}) = 1 - \frac{2y\hat{p} + 1}{y + \hat{p} + 1}$$

Here, 1 is added in numerator and denominator to ensure that the function is not undefined in edge case scenarios such as when y = $\hat{p}$ = 0.

4.4 Optimizers

Stochastic Gradient Descent (SGD) is an implementation of gradient descent which approximates the real gradient of the loss function [12].
It is computed by taking into account all the training examples, with an approximated gradient which is calculated by iteratively taking a single training example at a time until it has gone through all training examples.
The word ‘stochastic‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration [13].

$$v^t = \mu \nabla L(w) + \alpha v^{t-1}$$

Here, v^{t} is the vector of updates at time t, the first factor is the gradient step and the second one the momentum.
In particular μ is a parameter called learning rate, which is controlling the size of the update steps along the gradient. Larger steps mean that the weights are changed more every iteration, so that they may reach their optimal value faster, but may also miss the exact optimum. Smaller steps mean that the weights are changed less every iteration, so it may take more epochs to reach their optimal value, but they are less likely to miss optima of the loss function.
The momentum corrects the update at time t with a fraction of the update at time t-1. Indeed it also accumulates the gradient of the past steps to determine the direction to go.
We use Nesterov momentum, which is a variant of the previous technique. The difference is just the position at which the gradient is computed: before or after the momentum step.
Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal scaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients [14].
The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.
Indeed, while Stochastic gradient descent maintains a single learning rate for all weight updates and the learning rate does not change during training, with Adam optimizer a learning rate is maintained for each network weight and separately adapted as learning unfolds.
Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Where v_t and m_t are the exponentially decaying average of past squared gradients and the exponentially decaying average of past gradients respectively.

4.5 HyperParameter Tuning

Before than fitting the U-net model with the entire dataset we try several combinations of hyper parameter in order to find the ones adapting better to our problem.
What we are going to study is how much the performances of the model change using different kind of loss functions, optimizers and number of layers of the net.
As explained in Section 4.3 we try Categorical Crossentropy, Focal and Dice Loss. Regarding the optimizers we train models using Adam optimizer and Stocastic Gradient Descent (SGD). Lastly we try to change the number of layers of our U-Net model observing how much it affects the performances.
Due to our limited amount of resources we decide to execute the hyperparameter tuning on a very small subset of the dataset. Precisely we extract randomly a set of 1024 images from the training set to train our models and a set of 256 images as validation set.
We train our models using a fixed batch size (16) and a fixed number of epochs (5). In Table 1 are reported the results obtained with the metrics shown in Section 3.

Hyper Parameters Tuning Results - Training Set				
Loss	Optimizer	Mean IoU	Precision	Recall
Cce + Dice	Adam	0.41	0.51	0.64
Focal	Adam	0.22	0.23	0.61
Cce	Adam	0.13	0.23	0.57
Cce + Dice	Sgd	0.11	0.25	0.58
Focal	Sgd	0.10	0.23	0.57
Cce	Sgd	0.10	0.22	0.57

Table 1: Hyper Parameters Tuning Result on small Training Set

From the table above (Table 1 we can observe that the best performance using "Adam" optimizer and "Cce+Dice" loss. The high value obtained for Mean IoU is the most significant one. The Recall value is in all the cases much higher than the Precision one. This is probably due to an unbalanced distribution of the categories. The table below (Table 2) reports the performances of the model on the extracted validation set.

Hyper Parameters Tuning Results - Validation Set				
Loss	Optimizer	Mean IoU	Precision	Recall
Cce + Dice	Adam	0.47	0.75	0.73
Focal	Adam	0.46	0.82	0.61
Cce	Adam	0.46	0.77	0.61
Cce + Dice	Sgd	0.46	0.78	0.31
Focal	Sgd	0.46	0.78	0.08
Cce	Sgd	0.46	0.77	0.56

Table 2: Hyper Parameters Tuning Result on small Validation Set

Differently from the training results the performances on the validation set differ less each other. Anyway we decide to use as optimizer "Adam" and as loss function "Categorical Crossentropy + Dice Loss". We choose this approach due the higher MeanIou and because precision and recall are more balanced in this case.

Another experiment made deals with the number of layers of our model. We try to change it and analyze what varies in order to understand which is the best number of layers for our model.
Ronneberger et al. [7] suggests to implement the U-net model with 23 convolutional layers. In its paper Ronneberger images have size 256x256 . Following this approach he works on image 8x8 in the last layer. On the contrary we work on images 128x128, consequently if we implement 23 layers our last layer will deal with images of size 4x4.
We have tried to reduce the number of layers from 23 to 18 (one step less in the contractive and in the extractive path) and to increase it to 28. The results are obtained, as in the experiments shown before, training on 1024 images and validating on a subset of 256 images. Table 4 reports the results.

	18 Layers	23 Layers	28 Layers
Mean Iou	0.46	0.47	0.46
Precision	0.72	0.75	0.74
Recall	0.69	0.73	0.65

Table 3: Number of Layers Tuning

The higher scores are obtained by using a 23 layers model. The 18 layers and the 28 layers model are provide similar scores. With 28 layers we have an higher precision, with the 18 layers we obtain an higher recall. In conclusion we will implement our model with 23 layers as *Ronneberger et al.* [7] suggests.

The last experiment made deals with the comparison two approaches to get masks (Section 2.4). We trained two models with "Adam" optimizer, "Categorical CrossEntropy" loss function and U-net architecture with 23 layers for 5 epochs.

	One Hot Encoding	Mask with One Layer
Mean Iou	0.47	0.39
Precision	0.75	0.32
Recall	0.73	0.33

Table 4: Number of Layers Tuning

The scores are much higher in the case of One Hot Encoding computed masks. This is probably due to a clearer representation of the mask and a better approach to deal with overlapping.

4.6 Model Training

The results shown in Section 4.5 are used in order to train our model. To be more precise we train the model on the entire training set (24092 images and masks) with the following parameters:

- Number of epochs: 20
- Batch Size: 300
- Steps per epoch: 80 *(Dataset Size / Batch Size)*
- optimizer: ”Adam”
- loss: ”Categorical Crossentropy + Dice Loss”

In order to avoid overfitting ”Early Stopping” callback is used. ”Early Stopping” stops training when a monitored metric or loss has stopped improving [15]. We decide to set ”loss” as parameter to monitor. In addition to that we set the Early Stopping parameter patience to 10, where patience is the number of epochs with no improvement after which training will be stopped.
Exploiting the ”keras” tool Tensorboard we have obtained the graphs shown in *Figure 8* reporting the loss and iou score behaviour along all the epochs.

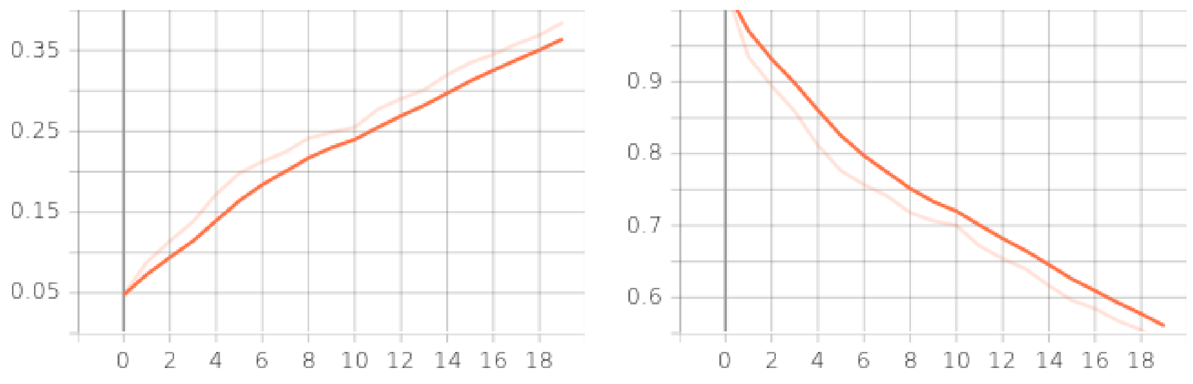

Figure 8: Loss and IoU score behaviour in 20 training epochs

Each graphs has two lines. The transparent one is the real behaviour, the other one is given by the original line smoothed of a 0.6 factor.
As we can see the two lines never become flat. Indeed all the 20 epochs are executed and early stopping is not applied.
Probably it should be a good idea to keep training the model without overfitting problems.
To conclude we use the model to predict the image segmentation masks of the test set. The result are reported in the next section.

5 Results Evaluation

In this section we are going to show the results obtained with the U-net model on the 1269 images test set. The hyper parameters are the ones found in the previous section. As explained in Section 3 we study the final scoring parameters IoU, precision and recall by averaging over all the values for all the masks.
Therefore we compute mean IoU, mean precision and mean recall:

	U-net Model
Mean Iou	0.45
Precision	0.72
Recall	0.65

Table 5: Model Scoring Parameters

The above results are very good with respect to the ones obtained on this dataset by the community. At the same time we have to take into account that we restrict the number of categories to 16. Due to this choice it is easier to obtain better results for what concern food recognition.

5.1 Binary Class Segmentation

We observe that for binary class images, i.e. all the images in which there is only one category of food, the model provides a good segmentation. Indeed the predicted mask often deviates a little bit from the true one.

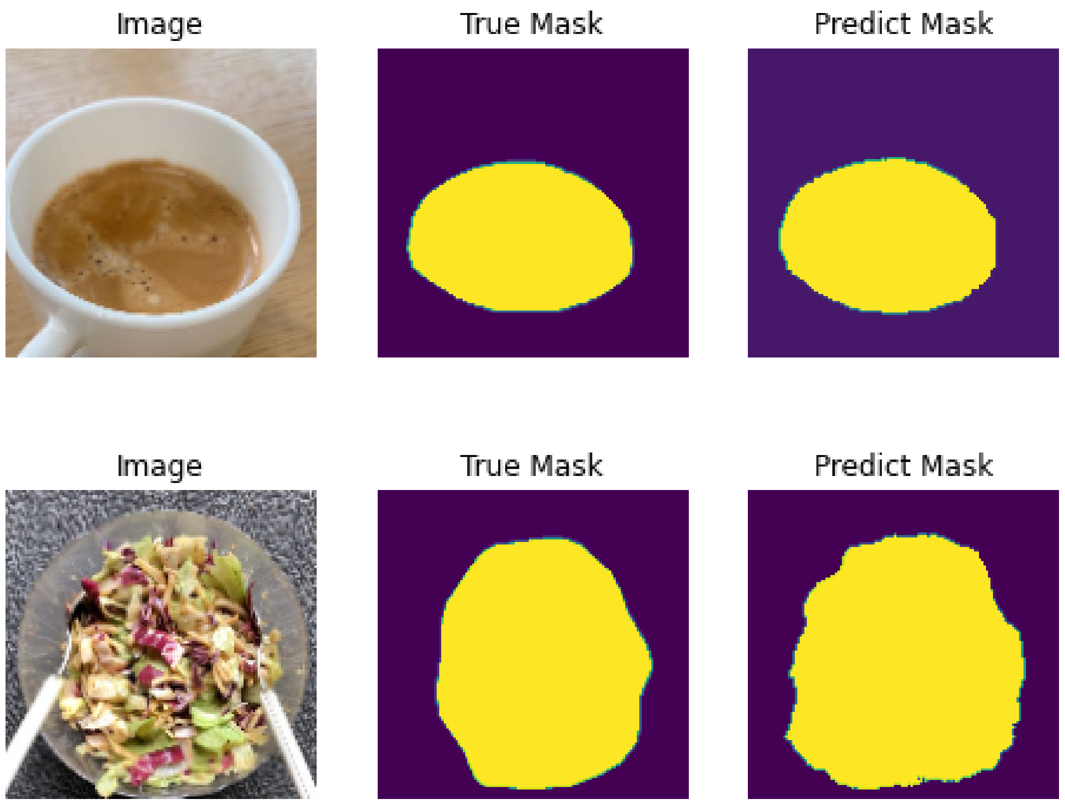

Figure 9: Examples of good binary class image segmentation

However there are some elements, e.g shadows, that can produce problems in the segmentation.

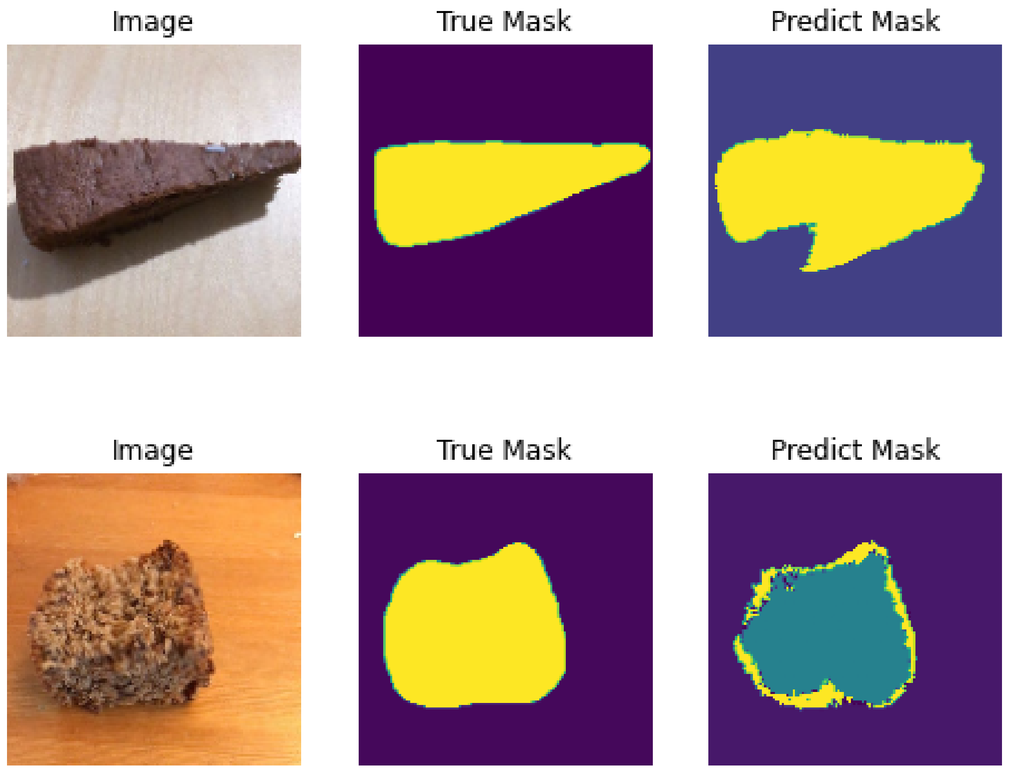

Figure 10: Examples of binary class image segmentation problems

In the upper image of *Figure 10* in the predicted mask there is the shadow of cake which is considered as a part of food element.
In the lower one the model is not able to distinguish perfectly the food category. Indeed, in the predicted mask there are more than one category, although the real image and the truth mask show only one food class.
Despite these problems, for binary class images the model provides good segmentation.

5.2 Multi-Class Segmentation

With regards to multi-class images the model is still able to perform a good segmentation.

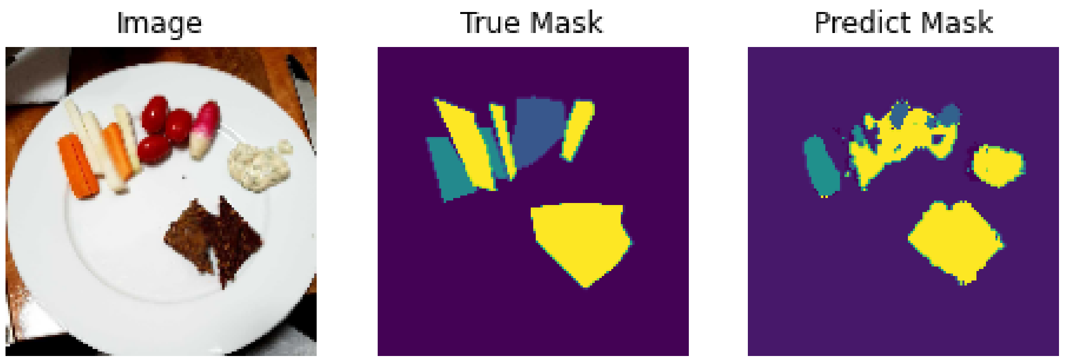

Figure 11: Example of multi-class segmentation

Therefore there are images difficult to segment with high precision. Indeed the model is not always able to recognize perfectly different categories, in particular where there is a significant area of overlap between two or more food classes.

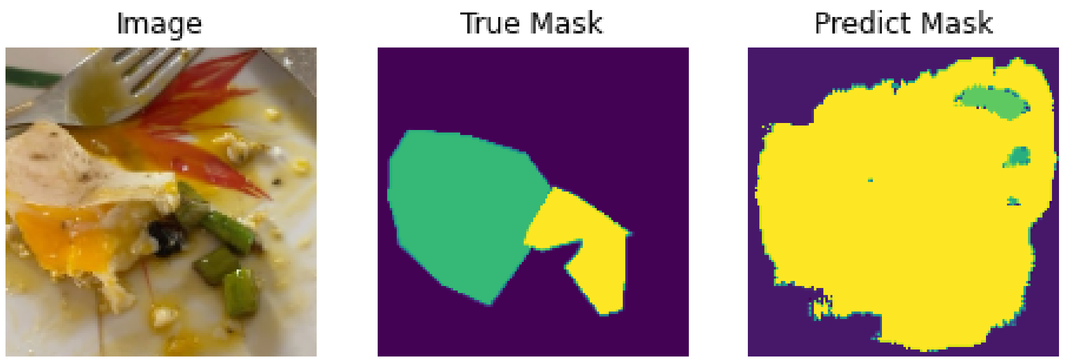

Figure 12: An examples of bad multi-class image segmentation

Despite these segmentation problems, there are some images in which the predicted mask appears more accurate than the true one. This is because annotations in the dataset are not completely exact.

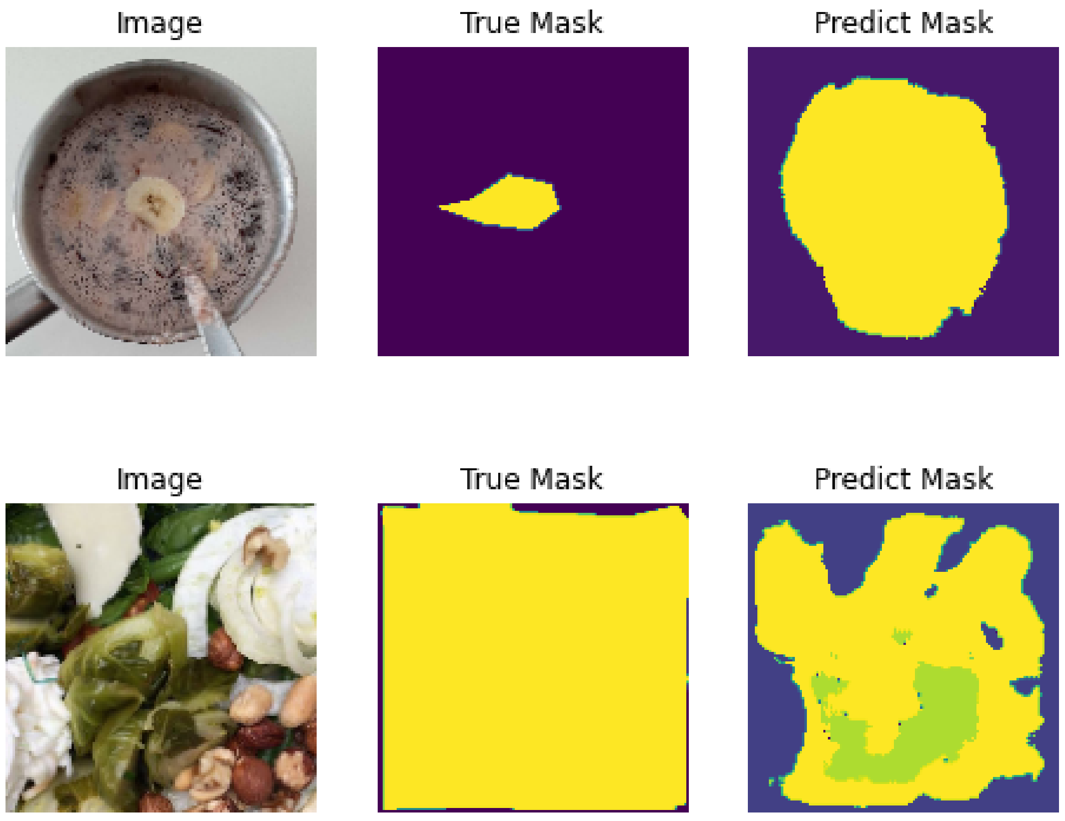

Figure 13: Examples bad true mask and good predicted mask

5.3 Class Recognition

The model trained is able to perform a good segmentation for binary and multi-class images and is able to recognize the different food classes.
There are images in which the model has problems to recognize all classes. In particular these images in which there are the different categories with respect to the 14 most frequent.

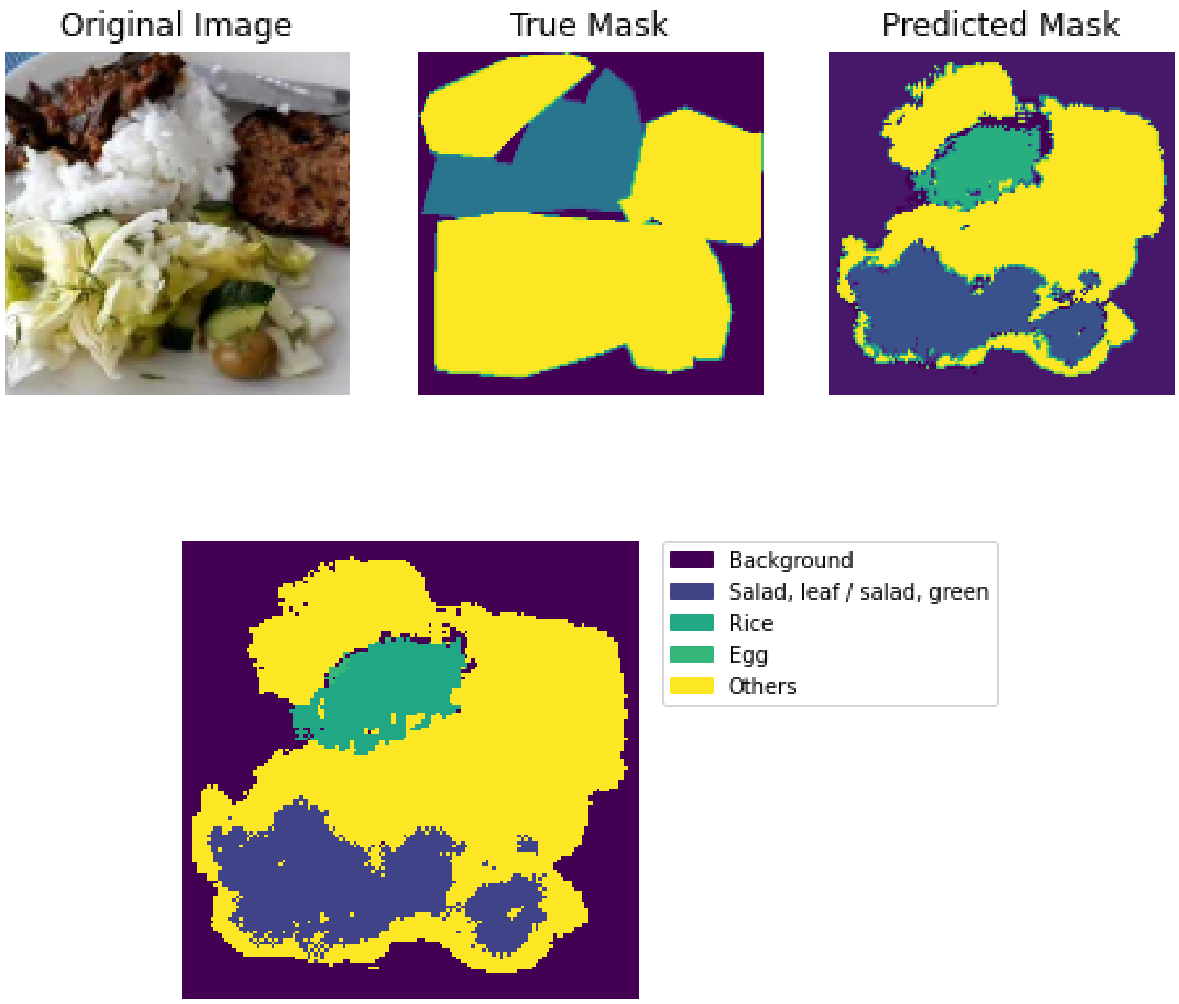

Figure 14: Recognizing food classes

As *Figure 14* shows the model has been able to recognize food classes. In this case it recognizes rice, salad leaf and egg. The last one is not present and it is a mistake but it is limited by a very small amount of pixels.

6 Conclusions

We can observe that the U-net model provides quite good results for the food image segmentation with respect to the evaluation criteria provided by "AICrowd Food Recognition Challenge Dataset" [2], as shown in the previous section. As already said we have to take into account that we worked on 14 categories grouping all the others as a unique category. Consequently it is easier to recognize the food class.
Anyway the model often estimates accurately the masks of binary class images and multi-class ones and it is able to recognize different food categories within the images provided by dataset.
However the model has some difficulties in the recognition of the food class, in particular in multi-class segmentation images. Indeed it makes some errors when there are different categories in an image with a large area of overlapping and in which food classes are mixed up.
Working on the dataset and printing some of the annotated masks, we have observed that some of them are not perfectly taken. This surely penalize our model perfomances.

We could improve our model perfomances by training it for an higher number of epochs. Indeed as show in *Figure 8* when we stop training the loss function is not flatten and the Early Stopping is not called.
Another possible improvement could be implementing data augmentation in order to provide more invariance to the network.
Lastly exploiting more powerfull hardware resources we could work an all the categories instead of working on just 14.

References

[1] "How u-net works?." /https://developers.arcgis.com/python/guide/how-unet-works/.

[2] "Aicrow food recognition challenge dataset." /https://www.aicrowd.com/challenges/food-recognition-challenge#datasets.

[3] P. Reddy, M. Nandini, E. Mamatha, K. Reddy, and A. Vishant, "Mask detection using machine learning techniques," in *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1468–1472, 2021.

[4] "Introduction to python generators." /https://realpython.com/introduction-to-python-generators/.

[5] "Python generators." /https://www.programiz.com/python-programming/generator.

[6] "Python generators." /https://towardsdatascience.com/paper-summary-u-net-convolutional-networks-for-biomedical-image-segmentation-13f4851ccc5e.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[8] "U-net architecture." /https://towardsdatascience.com/u-net-b229b32b4a71.

[9] "Activation functions." /https://deepai.org/machine-learning-glossary-and-terms/activation-function.

[10] "Categorical crossentropy." /https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

[11] S. Jadon, "A survey of loss functions for semantic segmentation," *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Oct 2020.

[12] "Sgd: Stochastic gradient descent." /https://peltarion.com/knowledge-center/documentation/modeling-view/run-a-model/optimizers/stochastic-gradient-descent.

[13] "Ml stochastic gradient descent." /https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[15] "Early stopping." /https://keras.io/api/callbacks/early_stopping/.