Image-based food recognition with U-Nets

Deep learning - Project 4

Alex Costanzino
alex.costanzino@studio.unibo.it
GitHub

Marco Costante
marco.costante@studio.unibo.it
GitHub

Academic year 2020-2021

Abstract

Recognizing food from images can be an extremely useful tool for a variety of use cases. For example, it could allow people to track their food intake by simply taking a picture of what they consume. Food tracking can be of personal interest, for example for fitness purposes, and can often be of medical relevance as well, such as in diabetological conditions or obesity control. Indeed, medical studies have for some time been interested in the food intake of study participants but had to rely on food frequency questionnaires that are known to be imprecise [15].

Image-based food recognition is a particularly challenging task. In a structured framework, in the past few years has made substantial progress thanks to advances in deep learning, but still remains a difficult problem to handle in a stand-alone fashion. More traditional computer vision approaches have achieved low classification accuracy and poor segmentation capability so far, while deep learning approaches enhanced the identification of food types and their ingredients.

In this report we propose an end-to-end approach through the use of U-Nets.

Contents

1 Introduction

The goal of the project is to train models that can be fed with images of food items and detect the individual food items present in them.

The dataset of food images is collected through the MyFoodRepo app [23] where numerous volunteer users provide images of their daily food intake in the context of a digital cohort named *Food & You*, while the challenge is raised up by *AIcrowd* [15], a crowdsourcing site that collects and sponsors several AI challenges.

This growing data set has been annotated, partially by hand and some other with classical algorithms, with respect to segmentation, classification (mapping the individual food items onto an ontology), and other physical properties, such as weight and volume estimation.

1.1 Criticality analysis

The contents of food dishes are typically deformable objects, including complex semantics, which makes the task of defining their structure very difficult.

There are some classes of food that are inherently hard to segment and classify [18], namely:

- Food with large inter-class similarity, such as drinks;
- Food with large intra-class diversity, such as rice and cereals;
- Incomplete dishes, that cause high variance in the shapes of the same class;
- Poorly taken photos, with respect to illumination or occlusions;
- Multiple food items;
- Unkown food.

Moreover, the high variance between the images is likely to cause numerical stability of the optimizers, with respect to the loss function.

1.2 Previous results

So far, most fruitful results have been achieved mainly by means of Mask-RCNN, SegNet and ENet [5].

At the moment, the best performance achieved on the leaderboard of *AIcrowd* is:

- $\text{AP}\{\text{IoU} > 0.5\} = 0.568$;
- $\text{AR}\{\text{IoU} > 0.5\} = 0.767$.

In this project we propose an enhanced version of the fully convolutional neural network U-Net, proposed by Ronneberger et al., in 2015 [3].

2 The data

Finding annotated food images is difficult, there are some databases with some annotations, but they tend to be limited in important ways. For example, most pictures that are provided by stock images or social networks tends to be exceptionally beautiful and tidy. However, algorithms need to work on real-world images, such as food that can be found in supermarket or canteen, to be purposefully used.

Dataset The provided dataset includes (Figure 1):

- A training set of 24120, as RGB food images, along with their corresponding 39328 annotations in MS-COCO format;
- A validation Set of 1269, as RGB food images, along with their corresponding 2053 annotations in MS-COCO format.

(a) Annotation done by hand

(b) Annotation done by algorithm

Figure 1: Annotation examples

MS-COCO format Microsoft's *Common Objects in Context* (COCO) is one of the most popular object detection dataset. It is widely used to benchmark the performance of computer vision methods. The COCO format is a specific JSON structure dictating how labels and metadata are saved for an image dataset, ensuring a fast and efficient compression of image features [2]. Indeed, the mask segmentation is stored as a list of vertexes of a polygon. The API itself provides a routine to convert them into a binary matrix.

2.1 Preprocessing

At beginning, while working on the dataset, we encountered several problems:

- Some annotations were inaccurate, totally wrong or even empty. This was due to the fact that most of data were gathered by users, that sometimes provided very poor segmentations, while some other times, the annotation was even missed during the acquisition process. Also, in some cases the classification was wrong;
- When there was an overlap between food, some annotations took into account the overlap, while some others did not, on the basis of the user's preference;
- Our machines were extremely limited in capacity, so we had to rely on *Google Colaboratory* during the training process, that provided a Dual Core Nvidia K80 GPU and a 12 Gb RAM . Nevertheless, the available resources was still restrictive, due to the limited GPU run-time assignment provided by the platform.

To overcome these problems we decided to consider only a subset of categories. Using the IoU metric as measure of overlap, we selected some categories that presented the issue, in order to treat this peculiar case as well. The other categories were chosen on the basis of their relative frequencies (Figure 2).

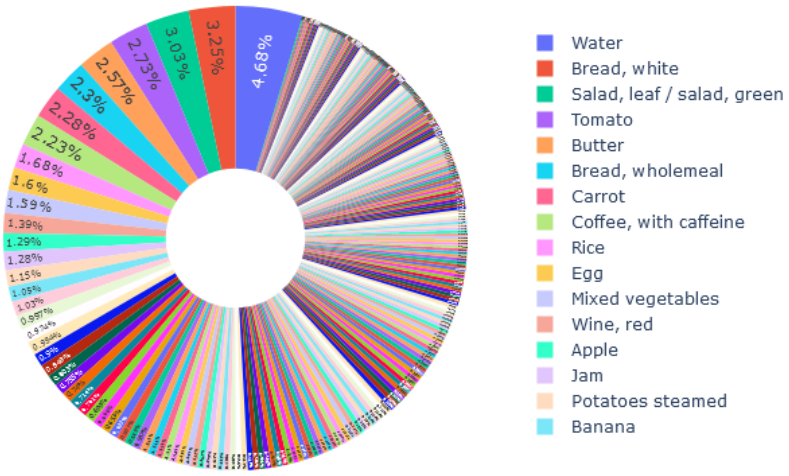


Figure 2: Relative frequencies of the first 16 categories

Moreover, we decided to resize the images to a common size. As first instance we tried to resize them to a median value, namely 480×480, in order to mitigate the downsamples

and upsamples. Due to the complexity of the chosen model we had to further resize them to 128×128.

One-hot encoding To take into account any possible overlap we decided to split the segmentation annotations onto more channels: one channel for each category, a channel for the background and one for the discarded annotations, following a one-hot encoding (Figure 3), making sure that the type of variables was floating-point, in order to avoid any error in convolution computations.

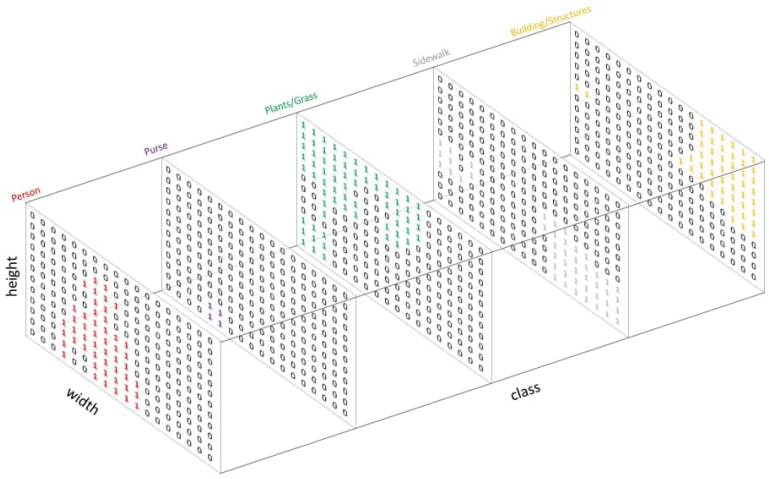


Figure 3: Example of one-hot encoding

Despite the dataset reduction, we ran into a memory-out error, due to RAM limitations, while loading the dataset. To solve the issue we decided to simply transform our image extractor function into a generator, that fetched batches of images [20].

3 U-Net

U-Nets so far have been mostly exploited for biomedical uses. Their architecture consists of a contracting path - the *encoder* - to capture context, and a symmetric expanding path - the *decoder* - that enables precise localization [3]. Such networks can be trained end-to-end from very few images, a scenario typical of the medical diagnostic, and outperformed, in 2015, the prior best methods.

3.1 Architecture

To be more specific, our architecture consists of the repeated application of bidimensional 3×3 unpadded convolutions, each followed by a *scaled exponential linear units* (SeLU), and a 2×2 max pooling operation with a stride of 2 for downsampling. At each downsampling step we double the number of feature channels (the number of filters).
Every step in the expansive path consists of an upsampling of the feature map, followed by a 2×2 up-convolution that halves the number of feature channels, a concatenation with the corresponding cropped feature map from the contracting path, and two 3×3 convolutions, each followed again by a SeLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer, a 1×1 convolution with a softmax activation function is used to map each feature vector to the desired number of classes. In total the network has 23 convolutional layers (Figure 4).
Each convolution/activation couple is interleaved with a batch normalization layer for regularization purposes.
The entire net is built *ex novo*, layer by layer, using the *Keras* backend [22] with the *TensorFLow* [24] API.
The main changes introduced by us are:

- The substitution of the dropout layer, widely used in literature, with the batch normalization layer, for regularization purposes;
- The substitution of the ReLU activations with SeLU activations;
- The substituion of the sigmoid activation with a softmax activation function, at the output of the neural network.

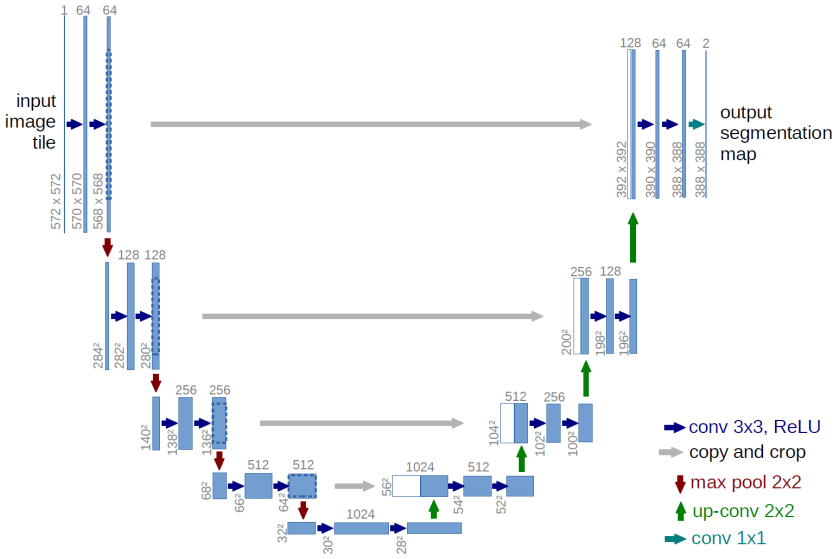


Figure 4: Original U-Net architecture from the paper

- Also, after some sessions of training, we decided to extend the original architecture of the U-Net, adding another encoder-decoder layer, to enhance the capacity of internal representation.

Batch normalization Is a method used to enhance neural networks, making them faster and more stable, through normalization of the layers inputs by re-centering and re-scaling the flowing data [1]. The reasons behind the effectiveness of batch normalization are still under discussion. It is believed that it can mitigate the problem of internal covariate shift, where parameter initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network. Some others argued that batch normalization does not actually reduce internal covariate shift, but rather smooths the loss function, which in turn improves the performance, as a form of regularization. However, at initialization, batch normalization induces severe gradient explosions in deep networks, which can be alleviated by skip connections, that is exactly the case of U-nets. Others assert that batch normalization, instead, achieves length-direction decoupling, and so it accelerates neural network.

We decided to substitute dropout with batch normalization due to stability problems that occurred during the training and, moreover, because batch normalization appears to be more efficient [17], especially in *TensorFlow* implementation. Also, it has a particular sinergy with the internal normalization provided by SeLU activation functions.

SeLU activation function *Scaled exponential linear units* (SELU, Figure 5) are activation functions that induce self-normalizing properties [7]. Similar to ReLUs, SELUs enable deep neural networks since there is no problem with vanishing gradients, nor exploding gradients. Furthermore, SELUs on their own learn faster and better than other activation functions, especially if they are combined with batch normalization [14].
The equation of the SeLU activation function is:

$$\text{SeLU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

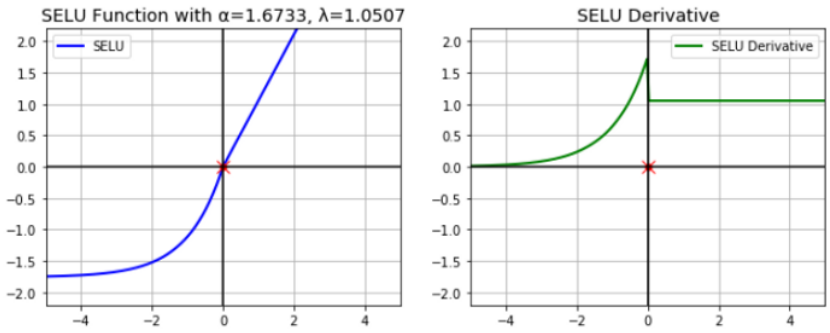


Figure 5: SeLU activation function and its first derivative

Softmax activation function The softmax function is a generalization of the logistic function, to multiple dimensions. It is often used as the last activation function of a neural network, to normalize the output of a network to a probability distribution over predicted output classes. The softmax function takes as input a vector x of n real numbers, and normalizes it into a probability distribution consisting of n probabilities proportional to the exponentials of the input numbers. Prior to the application softmax, some vector components could be negative, or greater than one, and also might not sum to 1. After applying softmax, each component will be in the interval $[0, 1]$, and the components will add up to 1, in order to be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.
The equation of the softmax activation function is:

$$\text{softmax}(j, x_1, \ldots, x_n) = \frac{e^{x_j}}{\sum_{j=1}^{n} e^{x_j}}$$

Concatenated skip connection Is a type of skip connection that seeks to reuse features by concatenating them to new layers, allowing more information to be retained from previous layers of the network. Long skip connections often are exploited in architectures that are symmetrical, where the spatial dimensionality is reduced in the encoder part and is gradually increased in the decoder part, as it happens in U-nets. In the decoder part, it's possible to increase the dimensionality of a feature map via transpose convolutional layers. The transposed convolution operation forms the same connectivity as the normal convolution but in the backward direction. By introducing skip connections in the encoder-decoder architecture, fine-grained details can be recovered in the prediction. Even though there is no theoretical justification, symmetrical long skip connections work very effectively in segmentation tasks [21].

4 Loss functions and metrics

4.1 Losses

During the design of the network several loss functions have been tested, most of them based on the paper *A survey of loss functions for semantic segmentation*, by S. Jadon [13]. Jadon makes a distinction between distribution-based losses and region-based losses: in our particular task, region-based losses seems to be the most promising.

4.1.1 Distribution-based losses

Categorical cross-entropy Is an extension of the *binary cross-entropy*, for multi-class segmentation tasks. It works well with skewed data and the *TensorFlow* implementation, `CategoricalCrossentropy()`, allows a weighting β of the classes, in order to mitigate any possible unbalance between them:

$$\mathcal{L}_{\text{CCE}}(Y_n, \hat{Y}_n) = -\frac{1}{N}\sum_{n=1}^{n}\left[\beta Y_n \log \hat{Y}_n + (1-\beta)(1-Y_n)\log\left(1-\hat{Y}_n\right)\right]$$

Categorical focal loss Is an extension of the *focal loss*, for multi-class segmentation tasks. It works best with highly imbalanced datasets and it downweights the contribution of easy samples, by means of a modulation factor, enabling the model to learn harder samples. We used an implementation from *Segmentation Models* library [25], `CategoricalFocalLoss()`:

$$\mathcal{L}_{\text{CFL}}(Y_n, \hat{Y}_n) = -\frac{1}{N}\sum_{n=1}^{n}\alpha Y_n (1-\hat{Y}_n)^{\gamma}\log \hat{Y}_n$$

4.1.2 Region-based losses

Dice loss Is inspired from *dice coefficient*, a widely used metric in computer vision community to calculate the similarity between two images, namely between the ground truth masks and the predicted masks. As Dice Coefficient is non-convex in nature, it has been modified to make it more tractable. Once again, we used an implementation from *Segmentation Models* library, `DiceLoss()`:

$$\mathcal{L}_{\text{DL}}(Y_n, \hat{Y}_n) = -\frac{1}{N}\sum_{n=1}^{n}\left[1 - \frac{2Y_n\hat{Y}_n + 1}{Y_n + \hat{Y}_n + 1}\right]$$

Jaccard distance Is based on the *intersection over union* metric, that is widely used in computer vision to measure the overlap between two images, namely between the ground truth masks and the predicted masks. In this case we made a custom implementation exploiting the *TensorFlow* backend, adding a regularization parameter λ:

$$\mathcal{L}_{\text{JD}}(Y_n, \hat{Y}_n) = -\frac{\lambda}{N} \sum_{n=1}^{n} \left[1 - \frac{Y_n \hat{Y}_n + \lambda}{Y_n + \hat{Y}_n - Y_n \hat{Y}_n + \lambda} \right]$$

4.1.3 Mixed approaches

Combination of categorical cross-entropy and dice loss, and categorical focal loss and dice loss, were also endeavoured. After some epochs they produced interesting results, but due to our computational limitations and the general complexity of the losses, these attempts were later relinquished.

4.2 Metrics

All metrics that we used were implemented by the *TensorFlow* API.

Accuracy Is the ratio of correctly predicted observation to the total observations:

$$\text{A} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Precision Is the ratio of correctly predicted positive observations to the total predicted positive observations predicted:

$$\text{P} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall Is the ratio of correctly predicted positive observations to the all observations in actual class:

$$\text{R} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Intersection over union Is the most popular evaluation metric for tasks such as segmentation, object detection and tracking. Object detection consists of two sub-tasks: localization, which is determining the location of an object in an image, and classification, which is assigning a class to that object.

$$\text{IoU}\big(Y, \hat{Y}\big) = \frac{Y \cap \hat{Y}}{Y \cup \hat{Y}}$$

The above metrics needs, of course, to be extended over all classes, including the background.

4.3 Evaluation criterion

For every known ground truth mask A_i, from the annotations of the validation set, a mask B_i is predicted by the neural network. Then, the $\mathrm{IoU}\big(A_i, B_i\big)$, a measure of the overall overlap between the true region and the proposed region, is first computed. Then, a true detection is considered, when there is at least half an overlap, namely when $\mathrm{IoU}\big(A_i, B_i\big) > 0.5$. This threshold is widely spread in the literature, but more restrictive criteria can be adopted.
Then we can define, for each mask, the following parameters:

- $\mathrm{Precision}\Big\{\mathrm{IoU}\big(A_i, B_i\big) > 0.5\Big\}$;
- $\mathrm{Recall}\Big\{\mathrm{IoU}\big(A_i, B_i\big) > 0.5\Big\}$.

The final scoring parameters are:

- $\mathrm{AP}\Big\{\mathrm{IoU}\big(A_i, B_i\big) > 0.5\Big\}$;
- $\mathrm{AR}\Big\{\mathrm{IoU}\big(A_i, B_i\big) > 0.5\Big\}$;

that are computed by averaging over all the precision and recall values for all known annotations in the ground truth, and predictions from the network.

5 Optmizers

5.1 Stochastic gradient descent

Stochastic gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent [9]. Is an iterative optimization technique that uses mini-batches of data to form an expectation of the gradient, rather than the full gradient using all available data.

For our project we used the *TensorFlow* API implementation, `SGD()`, with an high Nesterov momentum, which is particularly good for image-based segmentation problems.

5.2 Adam

The *Adam* optimization algorithm is an extension to the stochastic gradient descent algorithm, that has recently seen broader adoption for mosts deep learning applications, in particular in computer vision and natural language processing.

Classical stochastic gradient descent algorithm maintains a single learning rate for all weight updates, that does not change during training. With Adam, a learning rate is maintained for each network weight and separately adapted as learning proceeds.

Adam envelopes the benefits of both *AdaGrad* and *RMSProp*, other extensions of stochastic gradient descent algorithm: instead of adapting the parameter learning rates based on the average first moment, namely the mean, as in RMSProp, Adam also exploits the average of the second moments of the gradients, namely the uncentered variance [6]. In particular, the algorithm calculates an exponential moving average of the gradient and the squared gradient, while the parameters β_1 and β_2 control the decay rates of these moving averages. The initial value of the moving averages and the decay parameters values are close to 1, resulting in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before, and then calculating bias-corrected estimates.

Adam is computationally efficient, has little memory requirement, is invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data and parameters, that is exactly our case. For our project we used the *TensorFlow* API implementation, `Adam()`, in

particular with the *AMSGrad* variant, that rather than exponential average, uses the maximum of past squared gradients to update the parameters, enhancing the convergence [10].

5.3 Learning rate scheduling

During the train of the neural network we relied on a scheduler for the learning rate, provided by *TensorFlow* API, that adjust the size of the step each epoch of training. At the beginning of the training we needed an higher learning rate in order to allow the net to find a good spot in which intensify the search. Then, in later epochs we needed a lower step size to descent more precisely towards the optima. We tried to adopt two different types of learning rate heuristics.

Cosine decay Is a type of learning rate heuristic that starts with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again. The resetting of the learning rate acts like a simulated restart of the learning process with a reuse of good weights as the starting point of the restart (referred to as a warm restart, in contrast to a cold restart where a new set of small random numbers may be used as a starting point) [8]. The function of the periods of high learning rates is to prevent the optimizer from getting stuck in a local minima, while the function of the periods of low learning rates is to allow it to converge to a point as close as possible to the true global minima.

Exponential decay Is a pretty plain heuristic, where the learning rate is subject to exponential decay, namely it decreases at a rate proportional to its current value, like in mosts physics phenomena.

5.4 Gradient clipping

Another problem that we encountered during the training was the so called gradient explosion, hence, as first instance, we decide to adopt the *gradient clipping*, a technique that tackles exploding gradients. This technique involves clipping the derivatives of the loss function to have a given value if a gradient norm is less than a certain threshold [4].
Later in the design of the neural network, we also decided to add batch normalization and SeLU activations, nevertheless, we decided to keep the clipping since it seems to grant a certain stability to the loss during the training.

6 Training

We have tried several different combinations of loss functions, filters and initial learning rates. The most promising were:

Models	Loss	Epochs	Initial learning rate	No. of filters	No. of parameters
A	Focal loss	25	$2.5e-3$	32	$7,763,602+2,944$
B	Jaccard	18	$1.25e-4$	32	$7,766,546+2,944$
C	Jaccard	45	$1e-2$	16	$1,944,338+1,472$
D	Combined	5	$1e-3$	16	$1,944,338+1,472$
E	Focal loss	15	$2.5e-3$	16	$1,944,338+1,472$

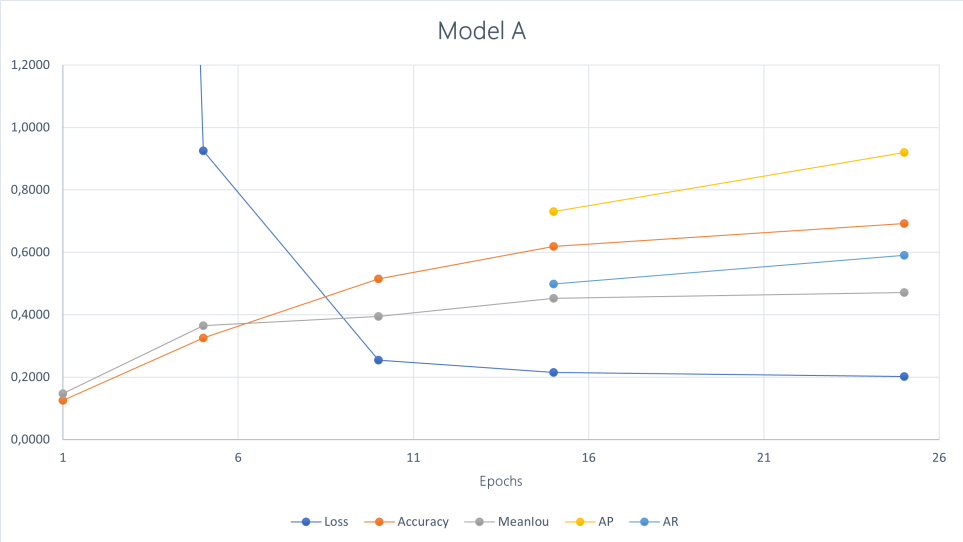
Moreover we made some trials with the "deep" U-Net model as well, even though the training was more problematic, due to the high number of parameters:

Models	Loss	Epochs	Initial learning rate	No. of filters	No. of parameters
dA	Jaccard	8	$1.5e-2$	32	$31,107,090+6,016$
dB	Focal loss	10	$1e-3$	32	$31,107,090+6,016$

Some models did not reach the desired IoU threshold, nevertheless, we are showing the results with a relaxed threshold, namely $\mathrm{IoU}\big(A_i, B_i\big) > 0.45$, since the results were interesting.

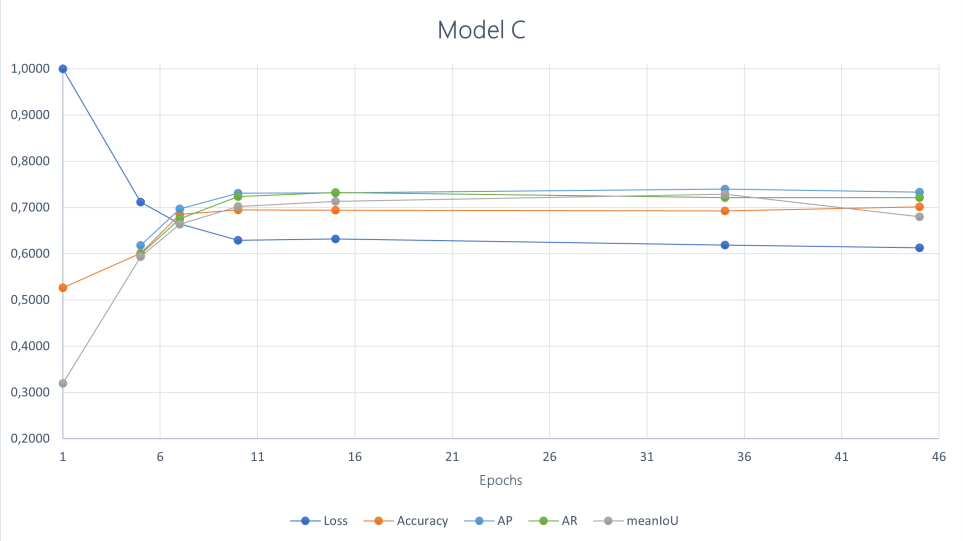
Model A Best result, with a relaxed threshold:

- $\mathrm{AP}\{\mathrm{IoU} > 0.45\} = 0.9218$;
- $\mathrm{AR}\{\mathrm{IoU} > 0.45\} = 0.5950$.



Model C Best result:

- $\mathrm{AP}\{\mathrm{IoU} > 0.5\} = 0.7332$;
- $\mathrm{AR}\{\mathrm{IoU} > 0.5\} = 0.7217$.



Model dA Best result:

- AP$\{\text{IoU} > 0.5\} = 0.7464$;
- AR$\{\text{IoU} > 0.5\} = 0.7165$.

Model dB Best result, with a relaxed threshold:

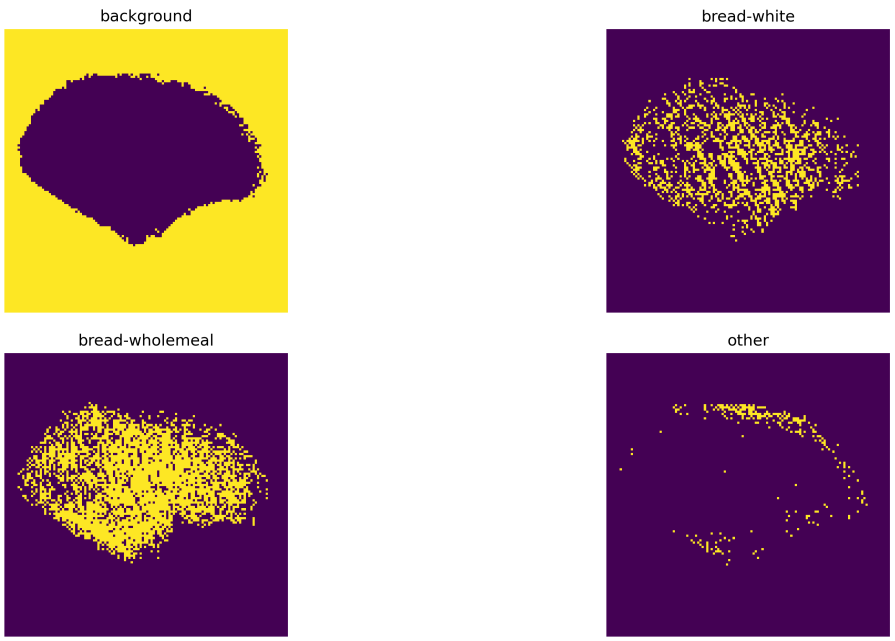
- AP$\{\text{IoU} > 0.45\} = 0.9242$;
- AR$\{\text{IoU} > 0.45\} = 0.5768$.

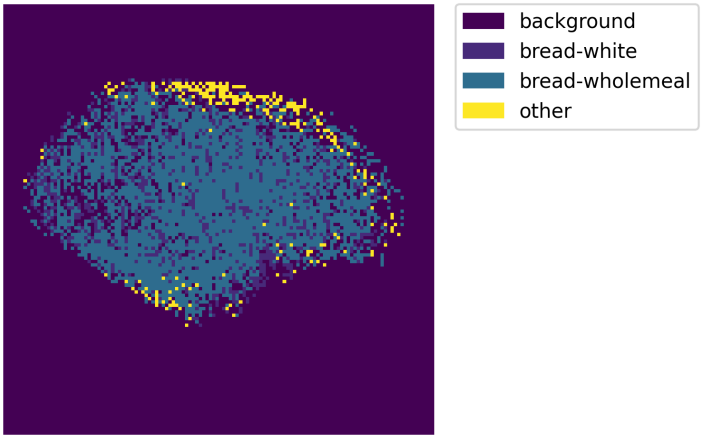
7 Results

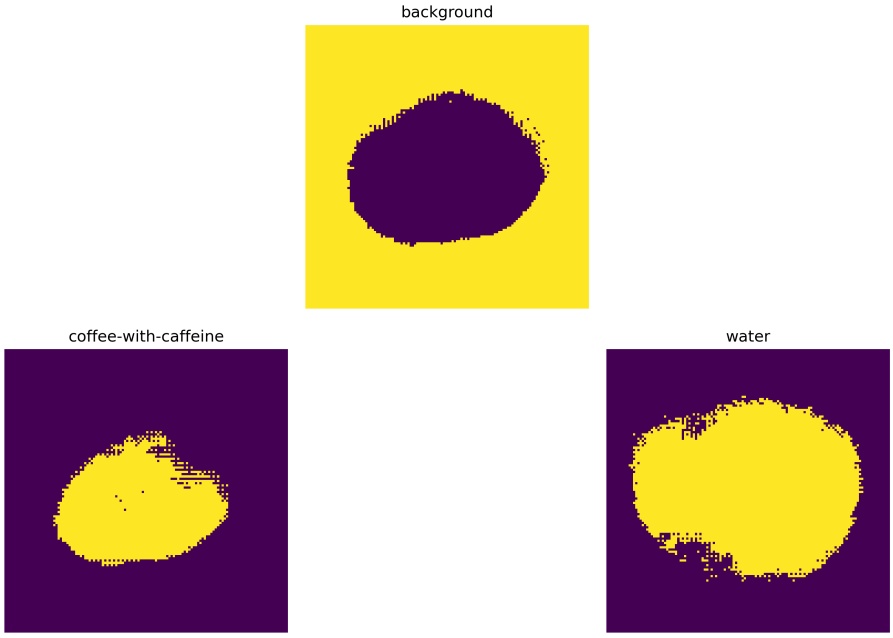
Despite the strong resize and the difficult tasks, the models provided some good results. However, examining the segmented masks, it's possible to observe the previously mentioned criticality.

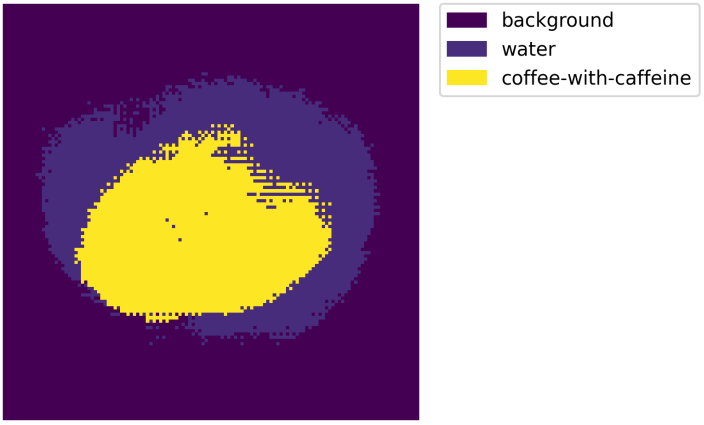
7.1 An example: high inter-class similarity

The models are not able to distinguish clearly between white and wholemeal kinds of bread, hence, in the masks some pixels are assigned to the first category and the others to the latter. The same happens with caffeine and water.

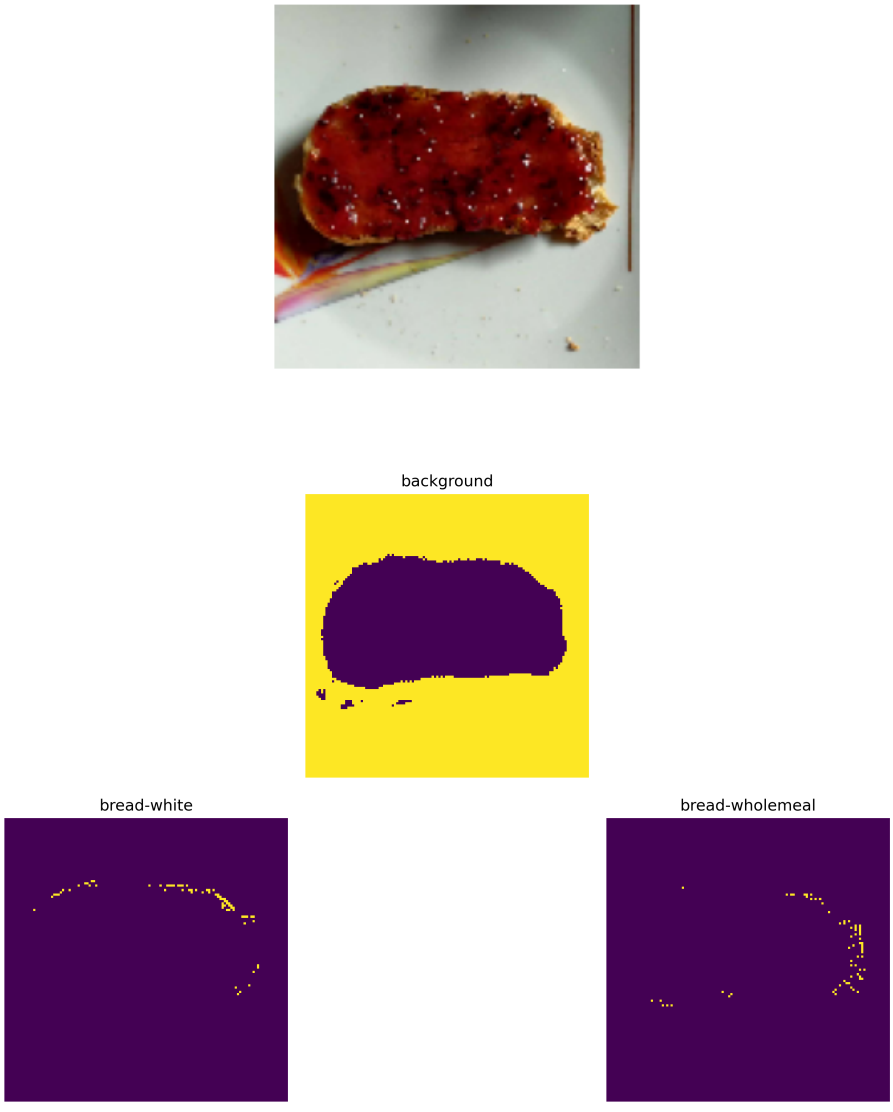


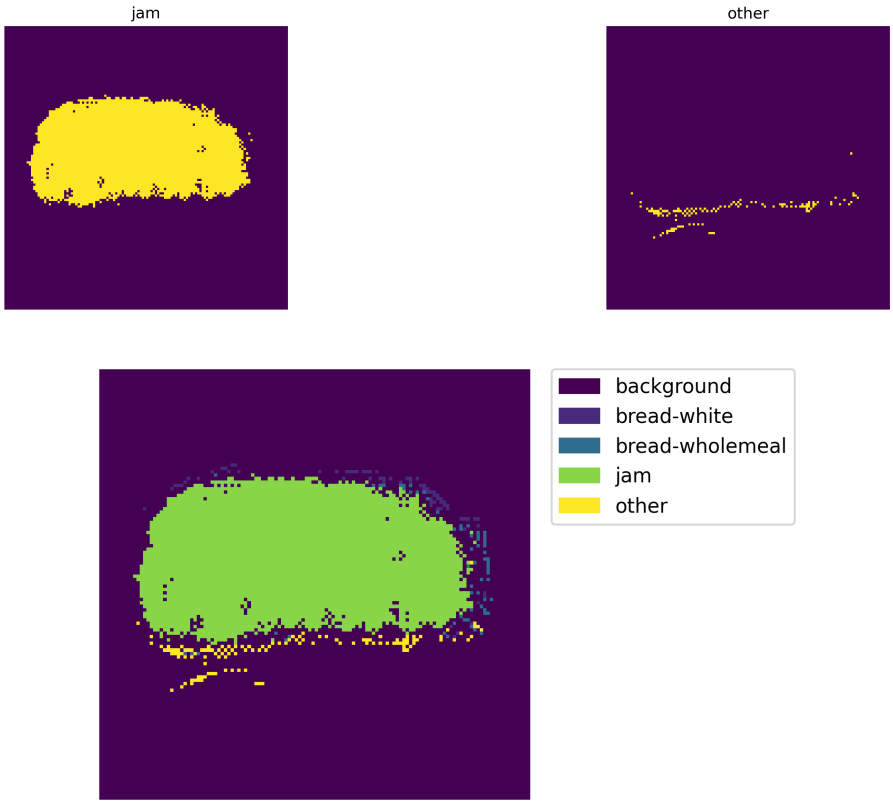

background
bread-white
bread-wholemeal
other


background
coffee-with-caffeine
water



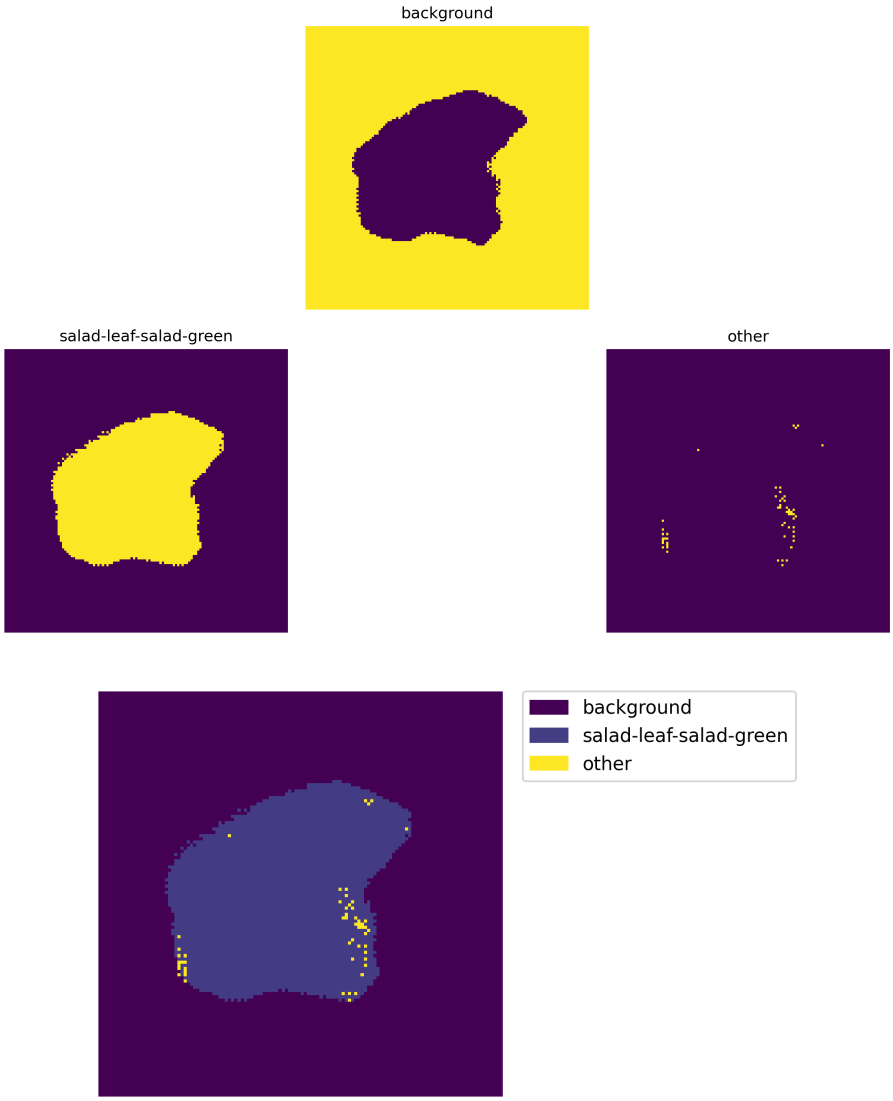
Nevertheless, with other classes instances, models still provide a good classification, even if they cannot distinguish the different types of bread.

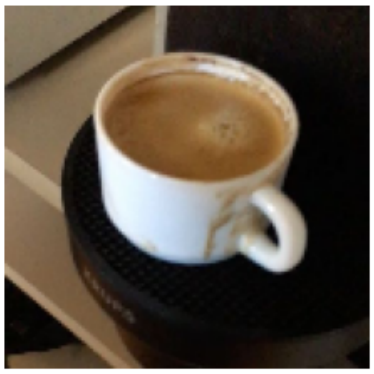


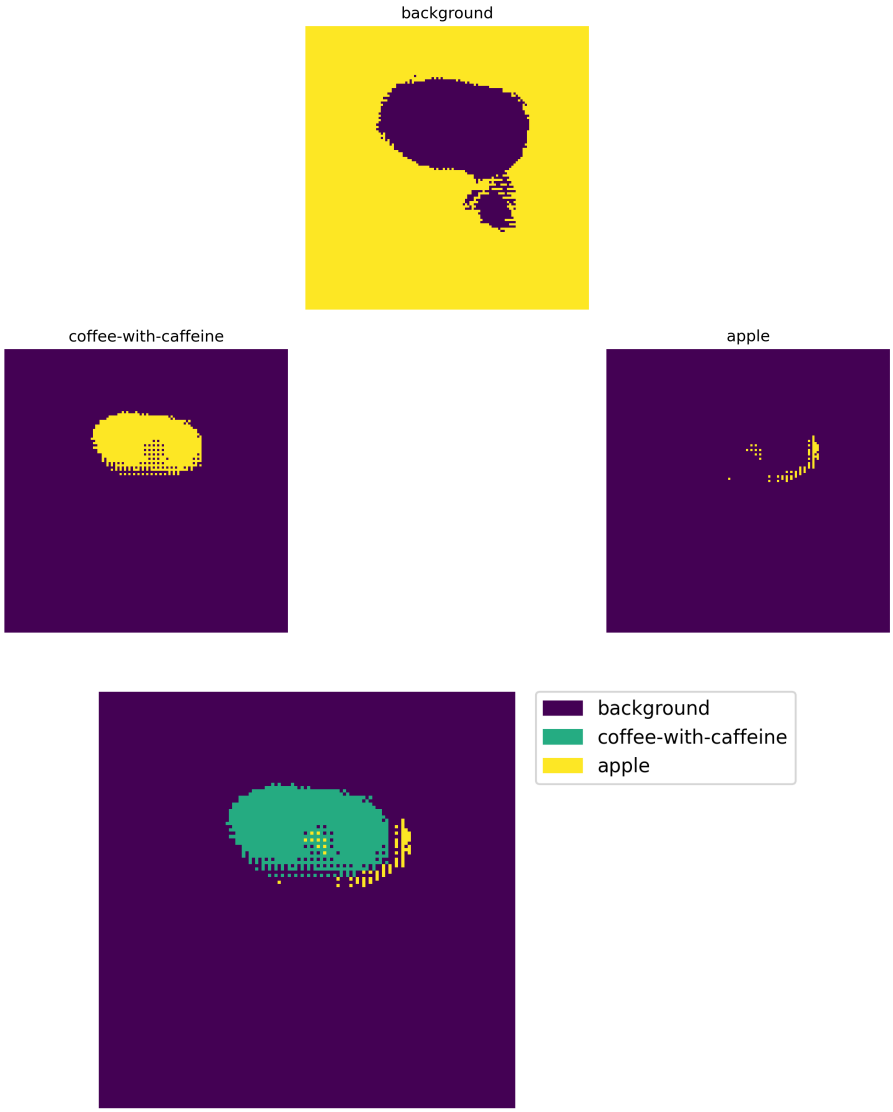


7.2 Good segmentations

Despite the previous problem, and some misclassified pixels, the models provide also good segmentations.


background
salad-leaf-salad-green
other
background
salad-leaf-salad-green
other





7.3 Conclusions

Despite being simpler than previous tried approaches, our approach with U-Nets still provides pretty good results, with respect to the evaluation criteria provided by the challenge, even though we are working with a reduced dataset.

The actual segmentation masks suffer of some issues, that can be simply be overcome with more training, especially with the use of "deep" U-net in sinergy with the combined loss, that should ensure both good classification and segmentation, with an higher capacity of internal representation.

7.4 Future developments

- Use *generalized intersection over union* [11] as evaluation criterion, in order to take into account well segmented but poor localized results;
- Use of data augmentation in order to provide more invariance to the network;
- Create a routine to prune or repair broken annotations, so that they do not affect the scores;
- Use transfer learning and ensemble approaches;
- Implement nested architectures, such as the one in U-Net++ [12];
- Redesigning skip connections to exploit multiscale features and overcome the issue of the search of the optimal depth [12].

References

[1] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167.

[2] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Adam Paszke et al. *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*. 2016. arXiv: 1606.02147.

[6] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[7] Günter Klambauer et al. *Self-Normalizing Neural Networks*. 2017. arXiv: 1706.02515.

[8] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983.

[9] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747.

[10] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. *On the Convergence of Adam and Beyond*. 2019. arXiv: 1904.09237.

[11] Hamid Rezatofighi et al. "Generalized Intersection over Union". In: (2019).

[12] Zongwei Zhou et al. "UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation". In: *IEEE Transactions on Medical Imaging* (2019).

[13] Shruti Jadon. "A survey of loss functions for semantic segmentation". In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)* (2020). DOI: 10.1109/cibcb48159.2020.9277638. URL: http://dx.doi.org/10.1109/CIBCB48159.2020.9277638.

[14] *A first Introduction to SELUs and why you should start using them as your Activation Functions*. URL: https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9.

[15] *AIcrowd*. URL: https://www.aicrowd.com/.

[16] *COCO - Common Objects in Context*. URL: https://cocodataset.org/.

[17] *Don't Use Dropout in Convolutional Networks*. URL: https://www.kdnuggets.com/2018/09/dropout-convolutional-networks.html/.

[18] *Food Image Recognition by Deep Learning*. URL: https://images.nvidia.com/content/APAC/events/ai-conference/resource/ai-for-research/FoodAI-Food-Image-Recognition-with-Deep-Learning.pdf/.

[19] *FOODAi*. URL: https://www.foodai.org/.

[20] *Implement generator in Keras*. URL: https://medium.com/@fromtheast/implement-fit-generator-in-keras-61aa2786ce98/.

[21] *Intuitive Explanation of Skip Connections in Deep Learning*. URL: https://theaisummer.com/skip-connections/.

[22] *Keras: the Python deep learning API*. URL: https://keras.io/.

[23] *MyFoodRepo*. URL: https://www.myfoodrepo.org/.

[24] *TensorFlow*. URL: https://www.tensorflow.org/.

[25] *Welcome to Segmentation Models's documentation!* URL: https://segmentation-models.readthedocs.io//.