Automatic point-of-interest image cropping via ensembled convolutionalization

Andrea Asperti and Pietro Battilana
University of Bologna
Department of informatics: Science and Engineering (DISI)

Abstract—Convolutionalization of discriminative neural networks, introduced also for segmentation purposes, is a simple technique allowing to generate *heat-maps* relative to the location of a given object in a larger image. In this article, we apply this technique to automatically crop images at their actual point of interest, fine tuning them with the final aim to improve the quality of a dataset. The use of an *ensemble* of fully convolutional nets sensibly reduce the risk of overfitting, resulting in reasonably accurate croppings. The methodology has been tested on a well known dataset, particularly renowned for containing badly centered and noisy images: the Food-101 dataset, composed of 101K images spread over 101 food categories. The quality of croppings can be testified by a sensible and uniform improvement ($3-5\%$) in the classification accuracy of classifiers, even external to the ensemble.

Keywords—Convolutionalization, crop, ensemble, food, Food-101, fully convolutional, heatmap, point-of-interest.

I. Introduction

As is it well known, there are two basic kinds of layers in a feed-forward neural network: *dense*, fully connected (FC) layers, following the structure of a traditional perceptron, and *convolutional* (CONV) layers [3], [8].

The only difference between the two kind of layers regards the connectivity of neurons, that in the case of convolutional layers is restricted to a local region of the input, and spatially replicated; the kernel of shared weights can then be regarded as a *filter* in the signal processing sense, that is *convolved* over the input. Apart from this distinction, the neurons in dense or convolutional layers share the same functionalities: they simply compute weighted sums of their inputs (dot products), allowing a simple conversion between FC and CONV layers. In particular, a FC layer with a two dimensional input of size $W \times H$ (typically flattened) can be simply understood as a convolutional layer with a large kernel of size $W \times H$. If the actual dimension of the input to this layer is $W \times H$, the kernel will be applied a single time (in "valid" mode), producing the same result as the original FC layer; however, if the input dimension is larger, the kernel will be convolved several times producing in output a *heatmap* of activations at different locations (see Fig.1).

The technique of transforming discriminative nets into fully convolutional ones (convolutionalization), was introduced for the first time in [12], with application to image segmentation. In this article, we explore instead its use for automatic cropping of the image to the expected area of the interest (see Figure 2).

The final aim is the improvement of image datasets with a more accurate location of the object category. This operation is usually expensive since it either requires human supervision, or highly specialized software: hence, having a general purpose and automatic technique is an important goal.

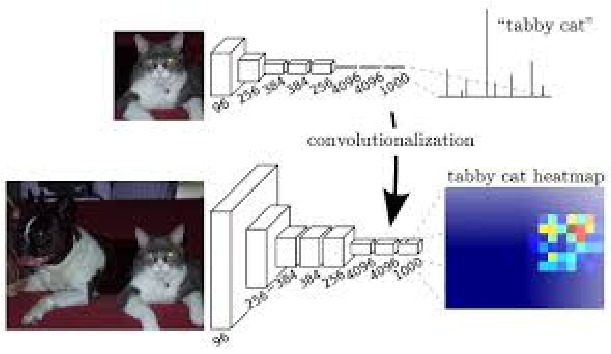


Fig. 1. Picture borrowed from [12]: transforming fully connected layers into convolution layers enables a classification net to output a heatmap

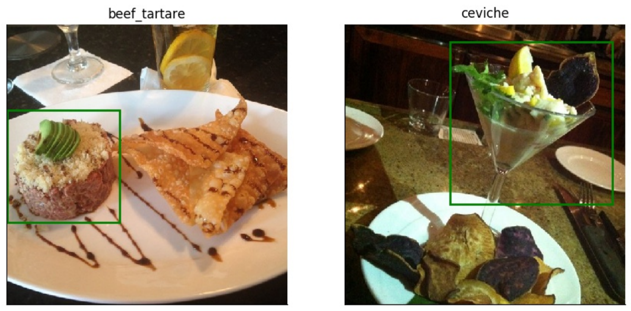


Fig. 2. Automatically computed crops relative to a given object category

Let we also observe that we are not forgetting or negating the relevance of noise for enhancing the robustness of neural networks, but while it is extremely easy to augment data with noise, the opposite operation, as any operation reducing entropy, is difficult and expensive. For instance, as a related example, we may recall the several works which have been devoted to the correct alignment of faces for the LFW dataset [7], based on funneling [5], deep funneling [6] or other commercial software.

We tested our cropping methodology on the Food-101 dataset, a challenging data set of 101 food categories, with 101.000 images. This dataset was introduced in [1] where the authors used Random Forests to mine Discriminant Components (RFDC) of the food images. The images of Food-101 are taken *in the wild* because they were collected from the social food guide foodspotting. To date, Food-101 remains the largest food dataset available. The dataset is also well known for containing noise (intense colors and some wrong labels) and badly centered images (see Fig. 2 and 3). Due to this reasons, the Food-101 dataset is a challenging dataset for our goal.

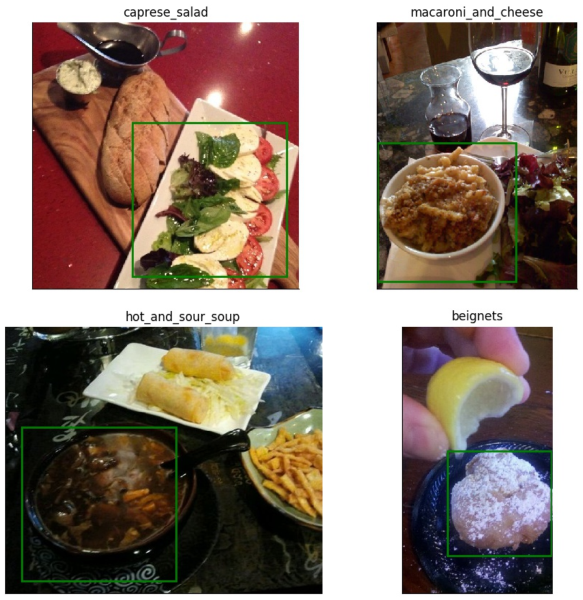


Fig. 3. More examples of images not centered on the given food category: in green the automatic crop

According to our experiments, relying on *a single* fully convolutive net (FCN) for cropping is easily prone to overfitting, and does not usually result in sensible improvements in the classification accuracy of other networks. On the other side, working with an *ensemble* of FCNs gives much more reliable results, resulting in an accuracy improvement of several percentage points even for classifiers not comprised in the ensemble (see Section IV).

The neural networks we considered are VGG16, VGG19 [13], InceptionV3 [15], InceptionResNet [14], and Xception [2], trained by transfer learning. We used VGG19 for testing, and the others for the ensembled convolutionalization.

The code is available on github[1]. Information about croppings and the weights of the trained networks can be retrieved from the project homepage[2].

The structure of the article is the following. In Section II we describe the networks we used and the way they have been trained; we also compare their classification accuracy with different works in the literature, testifying that we reached state-of-the-art results on the Food-101 dataset. Section III is devoted to the transformation of the networks in fully convolutional ones (FCNs): in particular we focus on the relation between the dimension of the input image and the dimension of the resulting heatmap, due to the need to compare heatmaps generated from different nets. More information on this argument is given in Appendix A. In Section IV, we present our methodology and discuss the results that have been achieved. Conclusions are given in Section V.

[1]https://github.com/pietrocarbo/ensembled_convolutionalization
[2]http://www.cs.unibo.it/~asperti/food101

II. Neural Network and the Food-101 dataset

We selected for our approach various deep CNN architectures famous for their performances at the ILSVRC competition. The weights were initialized using the ImageNet weights (*transfer learning*) and then every net was individually fine-tuned on the Food-101 dataset.

A. VGG

This is the oldest architecture used, but it's also the most straight-forward to optimize. The final 1000-neurons fully connected layer used to output the softmax classification for the 1000 category of ImageNet was replaced with a 101-neuron dense layer for the category of Food-101. Both the variants VGG16 (16 layers net) and VGG19 (19-layer net) were fine-tuned with a *two-pass* policy, that consists in just training, at first, the new final dense layer for various epochs, and then train all the layers of the net together.

B. Inception

It is a family of very deep and modular CNN that take the name from the Inception block (see Fig 4). This block apply several different-sized convolutional kernels and concatenate the resulting feature maps. To reduce the overfitting risk of the InceptionV3 model we replaced the final dense classification layer with three dense layer of 1024, 512, 101 neurons interleaved by *dropout* layers and *batch-normalization* layers. We also used the InceptionResNetV2 model variant that required a shorter training time thanks to the stability given by residual connections.

Given the depth of these architecture, the fine-tuning was carried out in steps. In order to clarify this process, let L be the set of layers allowed to be trained at training step s. The training step is declared over when the validation accuracy shows no improvement after different epochs. L is initialized at step $s = 0$ to include only the bottom dense layers; then, at each successive step, it is extended of a number of layers corresponding to the boundaries of the next Inception block not already included in L.

C. Xception

This model shares the same overall architecture of InceptionV3, improving on it. It takes advantage of new components as *residual connections*, and makes an extensive use of *depth separable convolutions*. In our experience, Xception seems to have a more stable training process and to be less prone to get stuck in local minima. Fine-tuning was carried out in steps as explained for the Inception nets.

D. Classification results

For evaluating performances of the networks we used the standard test set provided by the dataset Food-101, distinct from the train set used for training. In Table I, we compare the

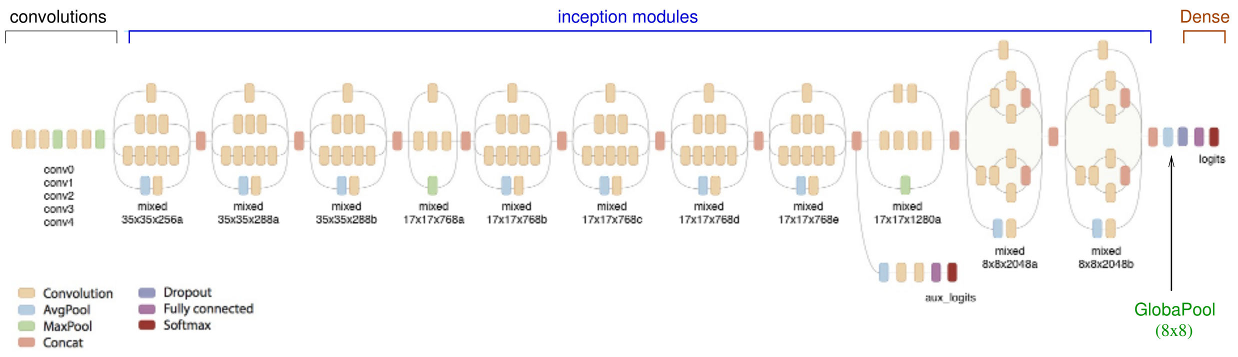


Fig. 4. The architecture of InceptionV3

TABLE I
ACCURACY COMPARISON ON FOOD-101 DATASET.

Model	Top-1 Accuracy	Top-5 Accuracy
RFDC [1]	50.76%	NA
AlexNet [1]	56.40%	NA
DCNN [16]	68.44%	NA%
FOOD-DCNN [16]	70.41%	NA%
FRSDA [9]	77.00%	94.00%
DeepFood [10]	77.40%	93.70%
VGG16 [our]	77.8%	94,0%
VGG19 [our]	78.3%	94.%
InceptionResNetV2 [our]	78.8%	94.3%
InceptionV3 [our]	79.1%	93.6%
Xception [our]	**80.7%**	**95.3%**
WISeR [11]	**90.27%**	**98.71%**

classification accuracy of the networks of Section II (suitably trained) with related works in the literature.

DCNN [16] stands for Deep Convolutional Neural Network: it is a modification of the AlexNet were the features extracted at the seventh convolutional layer were flattened into a one-dimensional array and then L2 normalized. The variant FOOD-DCNN was pretrained on a total of 2000 image categories, adding to the 1000 ILSVRC categories the most common 1000 food categories taken from the ImageNet database.

DeepFood [10] uses GoogLeNet as a starting point and slightly modifies the architecture of the Inception module. The adjustments consist in the additions of 1×1 convolutional layers that reduce the dimensionality of the input allowing to exploit deeper networks.

The model called "FRSDA" [9], which stands for "'Food Recognition System for Dietary Assessment", also uses an implementation of the GoogLeNet CNN with 22 layers modified using different kernel sizes and stride dimension.

The model presented by Martinel et al. [11] currently represents the state-of-the-art for the classification task on Food-101. The architecture is composed of two branches: the *wide residual branch* [17] encodes generic visual features of the food images while the *slice branch*, implemented by image-wide convolutional kernels, learn the vertical traits of particular food dishes. The output of the two branches gets fused via concatenation and then fed into two dense layers that emit the food classification.

The results in Table I prove that, apart for the highly specialized architecture of WISEeR, training was successful, allowing to attain state-of-the-art classification accuracies.

III. FULLY CONVOLUTIVE NETWORKS

We shall explain the transformation of a discriminative network into a fully convolutive one discussing the case of InceptionV3 [15], one the neural networks we used for our experiments.

The structure of the net is described in Fig. 4. Similarly to most networks conceived for image recognition, the network is composed by a first convolutional part, aimed to extract significant deep features from the image, followed by a sequence of dense layers, exploiting the features to synthesize the result (e.g an object category). In the case of InceptionV3, the convolutional part is composed by a few initial traditional convolutions followed by a long sequence of *inception modules*. Between the convolutional and the fully connected part, it has become traditional to insert an average *global* pooling layer. The advantage of having a global layer, is that the dimension of the output (that must have a fixed dimension, being fed as input to dense layers) only depends on the depth of channels, and is thus independent from the size of the input. In other words, you may pass as input to InceptionV3 (similarly to all networks of Section II) an arbitrarily large image: the global average layer will average each feature on a suitably sized area, spanning the whole input.

The ratio between the input dimension W and the dimension D of the average pooling essentially depends on the number of convolutions with non-unitarian strides between them. For all the networks in Section II we have 5 convolutions with stride 2, responsible for a total spatial reduction by a factor $2^5 = 32$. However, the precise relation is a bit more complex, due to the possible presence or absence of padding (see Appendix V). In particular, for each D there is an interval $[W, W+31]$ of input dimensions compatible with it (see Table II).

TABLE II
INPUT DIMENSION W W.R.T THE DIMENSION D OF THE GLOBAL AVERAGE LAYER; ALL DIMENSIONS IN THE INTERVAL $[W, W+31]$ ARE COMPATIBLE WITH D.

class.	VGG16	VGG19	InceptionV3	ResNet	Xception
W/D	32D	32D	32D+43	32D+43	32D - 25

When we convolutionalize the network, we must fix the dimension D of the averaging area *a priori*: in a sense, that will be the scale at which we are are looking for our object. The dimension could be arbitrary (even a single point), but if we trained the networks to recognize an object by averaging features on a given area, it looks natural to use the same area in the fully convolutional network (things would be different for segmentation purposes).

For instance, in the case of InceptionV3, an input of dimension $W = 299$ corresponds to an averaging area of dimension $D = 8$ ($299 = 32 * 8 + 43$). This means that the resulting FCN will behave as a convolution with kernel dimension 299 and stride 32.

In order to transform the net into a fully convolutional one, it is enough to transform the average max pooling into a usual, convolutive average pooling with kernel 8×8 (and stride 1), and similarly change all successive dense layers to convolutions with a kernel of dimension equal to their inputs (now appearing in channel position, so if channels are in third position, the kernels will typically have dimensions 1x1xd for a suitable d).

It is important to observe that the resulting net has exactly the same parameters of the original one. So, we may import the weights of the original discriminative net (suitably reshaped); moreover, if we apply the fully convolutional network (FCN) to an input of the same size expected from the original net we shall get exactly the same output (with two additional dimensions, since it is now an element of a 3-dimensional array). The nice property, however, is that we can now apply the FCN to an arbitrarily larger image, obtaining in output a heatmap for each category, warmer at the locations where matching is higher.

A. Overlapping heatmaps

In order to work with an ensemble of FCNs, we need to compare their generated heatmaps, that should hence have the same dimension, and correspond to a same sampling of the input space. The simplest solution is to work with FCNs with a same "kernel" dimension, say 299. Working with kernels with different dimensions would not only require resizing the image to the expected size for each different FCN, but (since the stride is the same) would also result in convolving the kernel at slightly different locations on the input image, as described in Figure 5.

According to the formula in Table II, we obtain the area dimensions summarized in Table III.

TABLE III
DIMENSION D OF THE AVERAGE FILTER COMPATIBLE WITH A FIXED KERNEL DIMENSION OF $W = 299$

classifier	VGG16	VGG19	IncetpionV3	ResNet	Xception
D	9	9	8	8	10

A marginal problem of this approach is that the VGG16 and VGG19 classifiers, that had been trained on their traditional dimension of 224, should be retrained on input images of dimension 299. The original weights still makes sense (the only difference is a slightly larger averaging area), so we

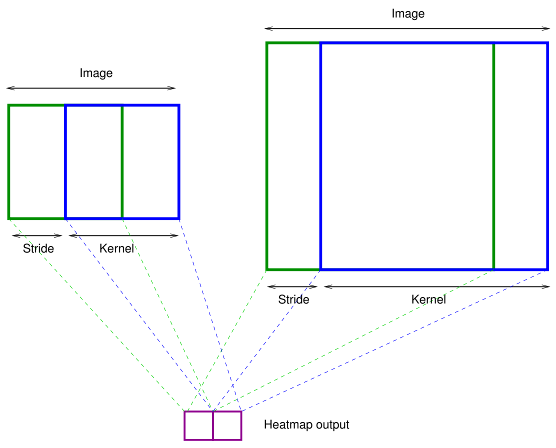


Fig. 5. The crops overlapping problem

obtain very good accuracy results in a few epochs of fine tuning.

IV. Methodology and results

Our goal is to detect where the point-of-interest of the image, in our case the food object, is located. The way to do this is by sliding the convolutional filter, represented by the whole trained CNN, across different areas of the image. Moreover, since we are interested to find objects at different dimensions, we rescale the input image by a fixed upsampling factor (we choosed 1.2, that is a traditional scale factor for object detection) up to a given maximum scale ($3:1$ in our implementation).

A. Crop selection

At each scale s, each FCN c generates a heatmap H_c^s where each cell is the output of the convolution on a given region of the input image, corresponding to the probabilities that the region contains the different object categories.

Since the heatmaps for the different FCNs have the same dimension (at a given scale s), we can just project on the inteded category ℓ, sum them together and take the cell with the highest value. The best crop $\langle x, y\rangle_s$ at scale s for the category ℓ is thus

$$\langle x, y\rangle_s = \operatorname*{argmax}_{i,j} \sum_{c \in \text{FCNs}} H_c^s[i, j, \ell] \qquad (1)$$

In this way we obtain a single "best" crop at each scale s; next, we need to compare crops at different scales to select one among them. We do this according to two citeria:

1) we start comparing the number of classifiers for which $H_c^s[i, j, \ell]$ is maximum among all categories (that is, the number of classifier that would correctly recognize the category ℓ if fed with the corresponding crop);
2) as secondary criterium, we use the sum of the confidence of each FCNs, that is $\sum_{c \in \text{FCNs}} H_c^s[x, y, \ell]$

B. Coordinate translation

Once selected the most promising heatmap element, we need to map its coordinates back into the input image space. Referring to 5 we are casting the heatmap output into the original image by following the blue dashed lines.

As already explained, we know that the heatmap elements correspond to squared surfaces in the input image. Given the most promising heatmap element we derive its squared surface in the input image by calculating the position of a vertex and the edge length.

This mapping depends on the *scale factor* by which the input image was rescaled before it was fed to the FCN. In particular, the edge length is calculated as:

$$\frac{train\_size}{scale\_factor} \tag{2}$$

where *train size* is the dimension of the images used to train the net. The position of the vertex also depends on the stride of the net (32 for all our nets) and it is calculated as:

$$\frac{32 \times hcoord}{scale\_factor} \tag{3}$$

where *hcoord* it is heatmap cell index. This formula is applied for both the heatmap cell indexes to obtain the (x, y) coordinates in the input image.

C. Results

Some examples of the resulting croppings are given in Fig. 6; more examples are reported in Appendix B.

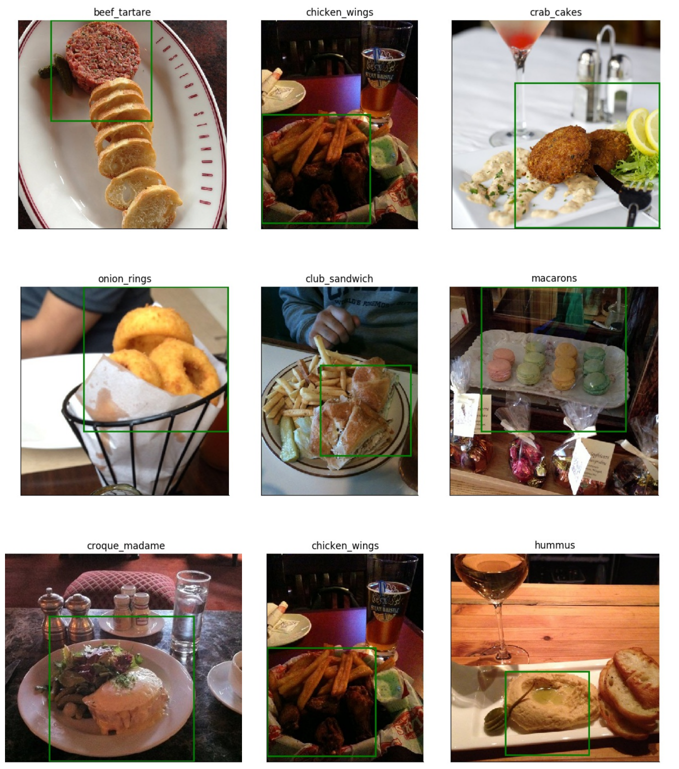


Fig. 6. Examples of automatic crops

Assessing the quality of croppings is not obvious, since we do not have at our disposal a ground truth locating the object in the image. To validate our approach in a quantitative way we used the trained CNNs as standalone classifiers on the original test set and on the *cropped* version.

TABLE IV
TOP-1 ACCURACY COMPARISON ON THE ORIGINAL AND ON THE *cropped* TEST SET.

Model	**Original**	**Cropped**
VGG16	77.8%	83.13%
VGG19	78.3%	81.20%
InceptionResNetV2	78.8%	85.23%
InceptionV3	79.1%	84.91%
Xception	80.7%	86.59%

The automatic cropping yields a remarkable accuracy boosts on all the models: this means that the ensemble is effectively removing noise from the images. The accuracy improvement is particularly significant in the case of VGG19, since this classifier was not comprised in the ensemble used to compute crops.

D. Identification of poor or missclassified samples

It is also possible to use our technique to find missclassified samples in the dataset. If, after convolving at several different scales, no classifier was able to find the food it is extremely likely that the sample had a wrong label. Some example are given in Figure 7. We identified several dozens of samples

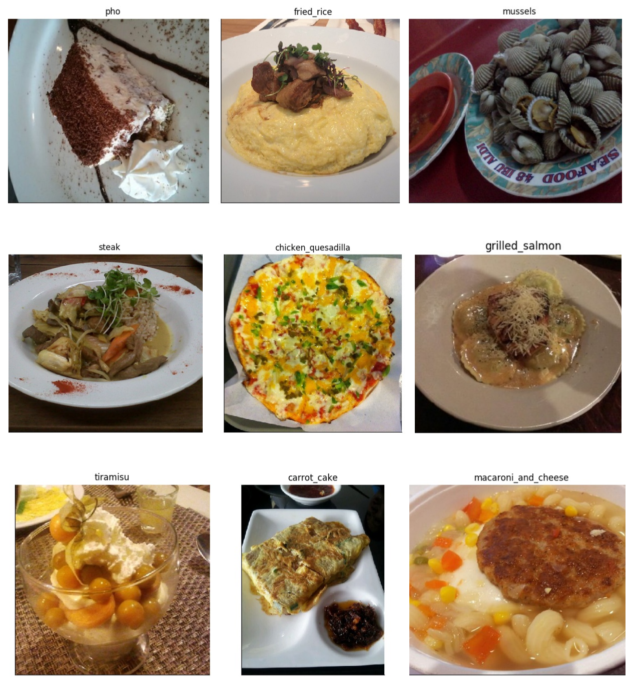


Fig. 7. Examples of missclassifications

with clearly wrong labels (relative to food we have confidence with), but *many* more looks questionable. We are preparing a *black list* that we shall publish on the web.

E. Problems

Although the crop computed by the application is almost always semantically meaningful, in the sense that correctly focus on a portion of the image pertaining to the given food, it does not always correspond to a neat or precise identification of the object's contours. The main problem is that, instead of focusing on the food *as a whole*, the crop may easily refer to some *detail* of the object (a color, a texture, a shape), of particular interest for its *discrimination* with respect to other categories, as exemplified in Figure 8. This is particularly evident when the "dish" is composed by a collection of different foods, like a cheese plate, or when it may be covered with very different kind of toppings, like a yogurt, a cake, or a quesadilla (see Figure 9). The problem is due to the fact that, as it is well know, CNNs "recognize" an object as a sum of small details, but have problems to understand their global, holistic structure (see e.g. [4]).

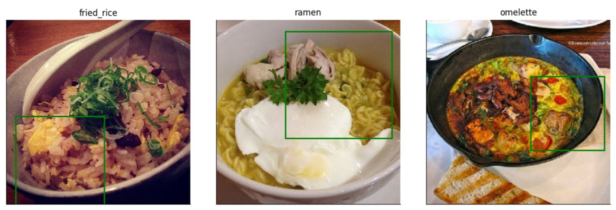


Fig. 8. Focusing on details

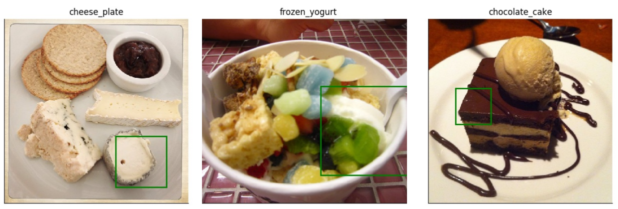


Fig. 9. Problems with composite food and toppings

V. CONCLUSIONS

In this article we showed how an *ensemble* of fully convolutional networks can be profitably used to crop an image to a given area of interest. The relevance of the technique is that it is fully automated and quite general: starting from a multi-category classification task, it can be applied to focus on a specific object-category in a large image. As for image cropping, it has many different applications, from the improvement of the underlying dataset, to an intelligent form of data-augmentation for training purposes.

According to our experiments, the parallel use of many FCNs is crucial to avoid overfitting the dataset on a given classifier; this is especially important for the Food-101 dataset, due to the moderate accuracy of the convolutional networks.

As explained in Section IV-E, many problems still remain to be solved, starting from an exceeding focus on object details, finding way to boost a more holistic view of the given object.

We also plan to integrate WISEeR [11] in our ensemble (provided we get hold of the networks weights), or to experiment with a similar architecture, in order to improve its classification accuracy.

APPENDIX A: KERNELS AND STRIDES

In this appendix, we investigate the relation between the input dimension of a typical CNN and the output of convolutions, up to the traditional average-pooling layer (excluded). As explained in Section III, this is essential to understand the kernel dimension and the stride of the resulting FCN.

We shall do the computation in the case of InceptionV3, but the reasoning is similar for other nets. We recall the structure of the net in Fig. 10.

The starting point of our analysis is the formula relating input size W, kernel size K, padding P, stride S and output dimension D for a given convolution:

$$\frac{W-K+P}{S}+1=D \tag{4}$$

We shall apply it backward, reconstructing W from D, that gives us:

$$W=(D-1)*S-P+K \tag{5}$$

In the case of InceptionV3, P is always zero (convolutions work with mode=valid), so we may forget it.

For instance, suppose to start with the avg-pooling layer with dimension $D = 8$ (we consider only one of the two dimensions). Going backward into the net, we discover that the first layer changing the size is a maxpooling layer with kernel $K = 3$ and stride $S = 2$, so the input dimension is $W = (8-1)*2+3 = 17$ (that is consistent with Fig. 4).

Most Inception modules do not change the size, up to another maxpooling layer with kernel $K = 3$ and stride $S = 2$, giving us the new input dimension $W = (17-1)*2+3 = 35$.

The initial convolutions are as follows (Keras code):

```
x = conv2d_bn(img_input, 32, 3, 3,
              strides=(2,2), padding='valid')
x = conv2d_bn(x, 32, 3, 3, padding='valid')
x = conv2d_bn(x, 64, 3, 3)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv2d_bn(x, 80, 1, 1, padding='valid')
x = conv2d_bn(x, 192, 3, 3, padding='valid')
x = MaxPooling2D((3, 3), strides=(2, 2))(x)
```

Using our usual formula, we then obtain the following sequence of dimensions (still going backwards):

$$\begin{array}{l}(35-1)*2+3=71\\(71-1)*1+3=73\\(73-1)*2+3=147\\(147-1)*1+3=149\\(149-1)*2+3=299\end{array} \tag{6}$$

Correctly, 299 is the expected input dimension for InceptionV3. Summing up, the formula relating the input dimension W to the output dimension D of the convolutive part is, for InceptionV3,

$$W=((((D-1)*2+2)*2+2)*2+4)*2+4)*2+3=32D+43 \tag{7}$$

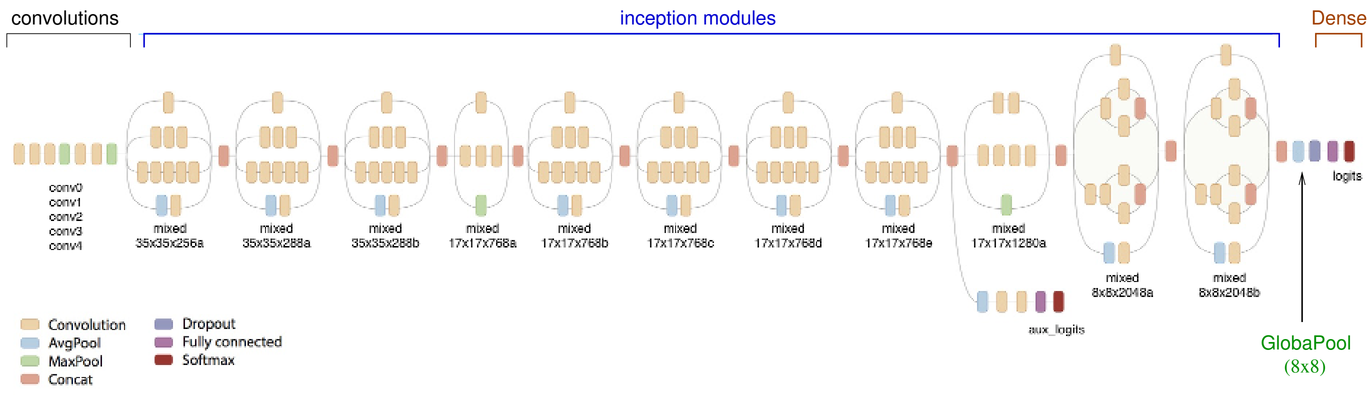


Fig. 10. The architecture of InceptionV3

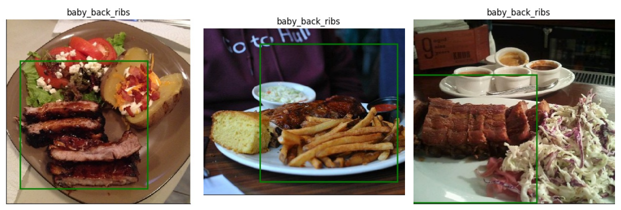
299 is just the "optimal" dimension, in the sense that it is the input dimension that would require minimal padding; however any dimension between 299 and 299+31 = 330 would produce an output compatible with a final pooling layer of dimension 8x8.

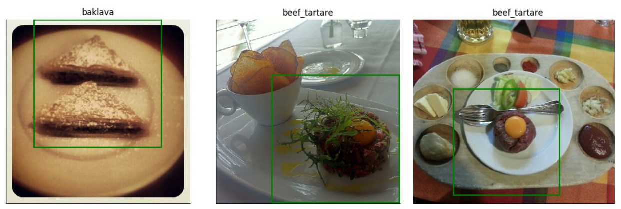
Repeating the same computation for the other architectures, we easily obtain the results in Table II, namely

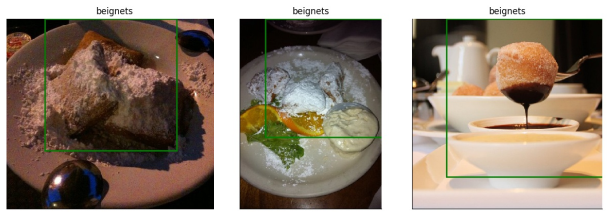
class.	VGG16	VGG19	InceptionV3	ResNet	Xception
W/D	32D	32D	32D+43	32D+43	32D - 25

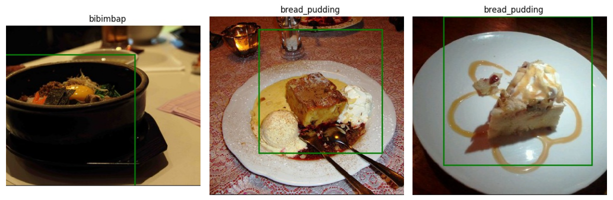
For each D, all input sizes in the interval $[W, W + 31]$ are compatible with it. Most of the networks work with convolutions with no padding (the so called *valid* mode), hence the "intended" input size (the one neglecting no part of the image) is the minimal one in the previous interval. Xception is a relevant exception, since its convolutions work in "same" modality. Hence, the best input size for a given D should be the maximal one, namely $32D + 6$ (the one requiring no padding). This seems to suggest that Xception should better work with input images of size $32 * 10 + 6 = 326$ (or 294, if you want to work with smaller images) instead of 299, as it is usually done. Questioned about this point, F.Choillet told us that the size of 299 was mainly choosed in view of constraints about the GPU's memory of the machines used to train the net.

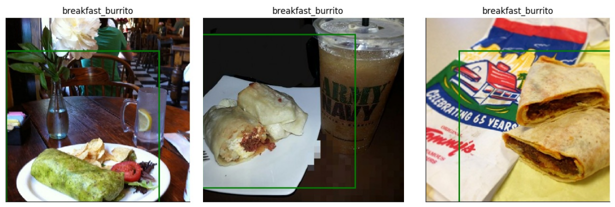
APPENDIX B: MORE EXAMPLES OF CROPPINGS

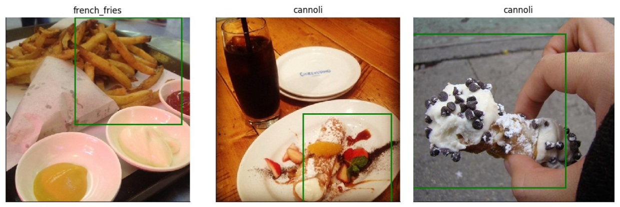












Many more examples available at http://www.cs.unibo.it/~asperti/food101

REFERENCES

[1] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.
[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.
[3] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[4] I. Goodfellow. Generative adversarial networks (gans). In *Thirtieth Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 5-10 2016. Tutorial*, 2016.

[5] Gary B. Huang, Vidit Jain, and Erik Learned-Miller. Unsupervised joint alignment of complex images. In *ICCV*, 2007.

[6] Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. Learning to align from scratch. In *NIPS*, 2012.

[7] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[8] Yann Lecun and Yoshua Bengio. *Convolutional networks for images, speech, and time-series*. MIT Press, 1995.

[9] C. Liu, Y. Cao, Y. Luo, G. Chen, V. Vokkarane, Y. Ma, S. Chen, and P. Hou. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, PP(99):1–1, To appear 2017.

[10] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, and Yunsheng Ma. Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. In *Inclusive Smart Cities and Digital Health - 14th International Conference on Smart Homes and Health Telematics, ICOST 2016, Wuhan, China, May 25-27, 2016. Proceedings*, volume 9677 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2016.

[11] Niki Martinel, Gian Luca Foresti, and Christian Micheloni. Wide-slice residual networks for food recognition. *CoRR*, abs/1612.06543, 2016.

[12] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, 2017.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[14] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[16] Keiji Yanai and Yoshiyuki Kawano. Food image recognition using deep convolutional network with pre-training and fine-tuning. In *2015 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops 2015, Turin, Italy, June 29 - July 3, 2015*, pages 1–6. IEEE Computer Society, 2015.

[17] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.