Deep Learning Techniques for Visual Food Recognition on a Mobile App

Michele De Bonis[1], Giuseppe Amato, Fabrizio Falchi, Claudio Gennaro, and Paolo Manghi

Institute of Information Science and Technologies of the National Research Council of Italy (ISTI-CNR) http://www.isti.cnr.it
{michele.debonis,giuseppe.amato,fabrizio.falchi,claudio.gennaro,paolo.manghi}@isti.cnr.it

Abstract. The paper provides an efficient solution to implement a mobile application for food recognition using Convolutional Neural Networks (CNNs). Different CNNs architectures have been trained and tested on two datasets available in literature and the best one in terms of accuracy has been chosen. Since our CNN runs on a mobile phone, efficiency measurements have also taken into account both in terms of memory and computational requirements. The mobile application has been implemented relying on RenderScript and the weights of every layer have been serialized in different files stored in the mobile phone memory. Extensive experiments have been carried out to choose the optimal configuration and tuning parameters.

Keywords: Convolutional Neural Network, Android App, Food Recognition

1 Introduction

The research in this paper exploits the use of Convolutional Neural Networks (CNN) for food recognition. A CNN is a type of artificial neural network which is based on a large collection of neural units (artificial neurons), loosely mimicking the way a biological brain solves problems with large clusters of biological neurons connected by axons.

Since CNNs are very complex and they need powerful hardware in order to get results in a reasonable time, they usually run on a device with high computational power in terms of CPU and GPU. The goal of our work was to design a food recognition system to be used by a mobile application, providing good recognition performance even when reducing the size of the neural network model to run on the limited computational and memory resources of a smartphone.

The idea of developing algorithms for food recognition is not new. Bossard et al. [2] present a method for food recognition based on a dataset of 101 categories of food. They used Random Forests to mine discriminant components of each class of food. Image classification is performed relying on Support Vector Machines (SVM). Wang et al. [12] present a method for food recognition based on the same classes of food in [2]. Exhaustive experiments of recipe recognition

using visual, textual information and fusion are carried out. Visual features are extracted using different methods, one of which is the OverFeat Convolutional Neural Network. Chen et al. [3] propose deep architectures for simultaneous learning of ingredient recognition and food categorization, by exploiting the mutual but also fuzzy relationship between them.

Amato et al. [1] propose a system called WorldFoodMap, which captures the stream of food photos from social media and, thanks to a CNN food image classifier, identifies the categories of food that people are sharing.

In our work, we developed an effective CNN for food recognition following these three directions:

- training from scratch various CNN architectures in order to select the most promising ones in terms of offered accuracy and network model size
- fine tuning, using food images, the most promising CNN architectures after a pre-training with the ILSVRC dataset [9]
- selecting the CNN with best accuracy/model size ratio in order to port it on the mobile phone and still guarantee high accuracy

In addition, we also performed a threshold analysis on the score of the prediction. This analysis helps to assess the confidence of the prediction made by the food recognition system.

2 Datasets

The experiments were executed on two datasets with 101 classes of very popular food types (i.e. 101 dishes). In literature, there are two versions of datasets with these 101 dishes using different sets of images: ETHZ Food101 [2] and UPMC Food101 [12] (see Figure 1).

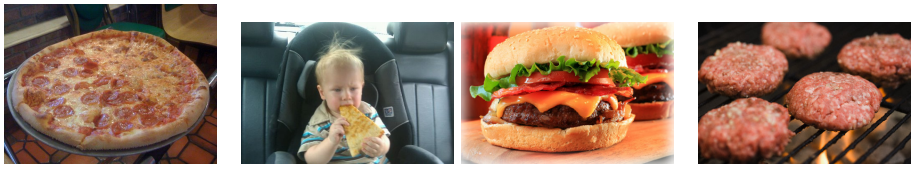
The way in which the images are collected to populate the two datasets are different, and this determines a difference in the number of images per class and in the type of the images.

We used two different datasets in order to have the possibility to perform cross-testing (e.g. train a CNN on a dataset and test on the other) to evaluate the so-called transfer learning (i.e. the ability to recognize images of another dataset).

The datasets present some noise images (i.e. images not representative of the class). This is due to the protocol used to collect them. In fact, as will be shown in next sections, images are collected by querying two different search engines and it might happen that the result of a query leads to wrong images.

Dataset	**number of classes**	**images per class**	**source**
UPMC	101	790-956	various
ETHZ	101	1000	specific

Table 1: Comparison of datasets



True Pizza Noised Pizza True Hamburger Noised Hamburger

Fig. 1: Example Images of ETHZ dataset (pizza) and UPMC dataset (hamburger).

We decided to format the images in both datasets in the same way. Moreover, we divided the datasets into three sets (training, validation and test set). The **testing set** of each dataset is exactly the same used in literature. This allows one to compare the results in terms of accuracy. UPMC dataset has a test set composed of 22,716 images, while the ETHZ dataset has a test set of 25,250 images. Remaining images have been divided into the **training** and the **validation set**. 25% of remaining images have been used for the validation, while 75% of them have been used for the training. Every single image has been formatted following the same specifications. Every image has been squashed to 256x256 pixels and encoded in PNG format in order to have no losses in the compression. During the database creation, the mean image of the dataset is computed. The mean image is subtracted from every input image in order to point out the significant regions of it.

3 Comparison of various CNN architectures

As we stated before, we used an approach based on Convolutional Neural Networks. A CNN is composed of a possibly large number of hidden layers, each of which performs mathematical computations on the input provided by the previous layer and produces an output that is given in input to the following layer. A CNN differs from classical neural networks for the presence of convolutional layers, which can better model and discern the spatial correlation of neighboring pixels than normal fully connected layers.

For a classification problem, the output of the CNN are the classes which the network has been trained on. The output layer is processed by a softmax function which "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1.

One of the objectives of our proposal is to run the CNN entirely on an Android mobile phone. To this purpose, the CNN has to be small and at the same time accurate and fast.

We tested the following CNN architectures: the AlexNet [6], the VGG-Net with 19 layers [10], the GoogLeNet [11], the SqueezeNet [5], the ResNet50 [4], the Binary Weighted Network and the XNOR-Net [8].

As a first step, we trained these CNNs from scratch using the ETHZ and UPMC datasets. This step aims to identify the most promising CNN for our purposes. We evaluated the CNNs relying on accuracy on the validation set (e.g. percentage of images of the validation set correctly classified).

We trained every CNN architecture by setting the same parameters. Generally, we set the number of training epochs to 60 but, in some cases, we used a different value depending on the accuracy trend of the CNN. We set the Batch Size to 128 for every scenario. Moreover, we chose a Learning Rate function with a step–down policy. This kind of policy consists in a step–down of 90% every 33% of epochs. If the number of epochs is 60, Learning Rate is fixed to 0.01 in first 20 epochs, to 0.001 in second 20 epochs and to 0.0001 in last 20 epochs. The Solver Type we choose is the Stochastic Gradient Descent (SGD).

CNN	**framework**	**model size**	**accuracy**	
			ETHZ	**UPMC**
ResNet-50	caffe	~80Mb	16%	20%
XNOR-Net	torch	~410Mb	25%	24%
BWN	torch	~380Mb	38%	34%
AlexNet	caffe	~230Mb	**40%**	**37%**
SqueezeNet	caffe	~3Mb	**41%**	**31%**
VGG-16	caffe	~590Mb	44%	37%
GoogLeNet	caffe	~40Mb	**56%**	**43%**

Table 2: Comparisons of various CNN architectures trained from scratch

Table 2 shows the result we obtained. As we can see, the most promising CNNs are the GoogLeNet (highest accuracy on both datasets and very small model size), the SqueezeNet (smallest model size and high accuracy on ETHZ dataset), and the AlexNet (good accuracy on both datasets and widely supported since it has many available implementations). Also the VGG-Net has a good accuracy on both datasets, but its model size is too big to fit in mobile memory, therefore it has been discarded.

In many cases, the training sets do not have a size sufficient to train a CNN with the required accuracy and generalization performance. This is also the case of the two training sets that we are using. In these cases, much higher performance can be obtained by pre-training the network (or use an already pre-trained network) on a different scenario, for which a very large training set is available, and then fine-tune the CNN on the scenario at hand with the small training set available.

In this respect, we used available pre-trained models of the most promising CNN architectures [1]. The models are those coming from training on the ILSCVRC [9] dataset (1,000 classes of images). For the fine-tuning process, we

[1] Pre-trained models can be found here: https://github.com/BVLC/caffe/wiki/Model-Zoo

used the same learning parameters used for training the networks from scratch. The weights of the last FC layer (which is mostly affected by the fine-tuning process) were initialized randomly, and the output was set to 101 classes (the ILSRVC dataset, used for pre-training has 1,000 classes).

An important parameter we set in the fine tuning is the Learning Rate Multiplier. It is multiplied by the Learning Rate and is set differently in each layer of the network to allow having different Learning Rates where needed. We set this parameter to 0.1 for the intermediate layers and to 1 in the last layer. This way the Learning Rate of the intermediate layers is 10 times smaller than the Learning Rate of the last layer.

CNN	**framework**	**model size**	**accuracy**	
			ETHZ	**UPMC**
AlexNet	caffe	~230Mb	59%	55%
SqueezeNet	caffe	~3Mb	**64%**	**55%**
GoogLeNet	caffe	~40Mb	**73%**	**63%**

Table 3: Comparisons of various CNN architectures fine tuned

Table 3 shows the result we obtained after fine-tuning. As we can see, the best CNN architectures in terms of model size and accuracy on the validation set are the SqueezeNet and the GoogLeNet.

4 Experiments

After the previous preliminary experiments, intended to identify the most promising CNN, we performed more detailed experiments focusing on the two selected CNN: the SqueezeNet and the GoogLeNet. We queried both CNNs with all the images in the testing set of both datasets and we logged the scores assigned by the CNN to each class. The most promising classes were assigned higher scores. We performed this process for both CNNs using both datasets performing cross-testing: we trained on one dataset and tested on the other in order to evaluate the transfer learning property.

We considered two metrics to evaluate:

- Accuracy Top-K: the percentage of images in the testing set classified in Top-K by the CNN;
- Distribution of Scores of Correct Class (Scc ICDF): the probability that the Scc of the images in the testing set will take a value greater than or equal to x (i.e. the Inverse Cumulative Distribution Function, considering the Scc values as a random variable);

Table 4 shows the Top1 Accuracy of the SqueezeNet. The best scenario is the one in which the CNN is trained and tested over ETHZ dataset (e.g. about 60%).

Train/Test	UPMC	ETHZ
UPMC	50.11%	37.51%
ETHZ	37.06%	59.71%

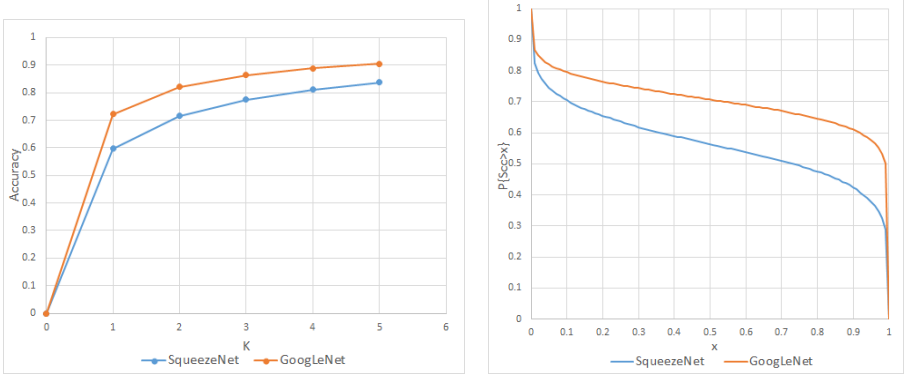
Table 4: SqueezeNet Cross Test: Top1 Accuracy

Table 5 shows the Top1 Accuracy of the GoogLeNet. Also in this case, the best scenario is the one in which the CNN is trained and tested over ETHZ dataset (e.g. about 60%).

Train/Test	UPMC	ETHZ
UPMC	60.95%	50.63%
ETHZ	46.17%	72.18%

Table 5: GoogLeNet Cross Test: Top1 Accuracy

Since the images in ETHZ dataset have strong selfie-style (images come from social networks and can contain also people eating food), we chose models coming from training on ETHZ dataset. We referred to ETHZ dataset also for the test for the same reason. This is because the CNN is going to be used by a mobile application, so the input image will be taken by the smartphone camera and will have strong selfie-style as well.



(a) Comparison of the Accuracies Top-K (b) Comparison of the Scc ICDFs

Fig. 2: Accuracy and Scc ICDF comparisons

Figure 2a shows the comparison of the Top-K accuracies of both CNN architectures. The GoogLeNet has the highest accuracy in general, but the difference between the two accuracies is decreasing over K. This means that, when

K is large, there is no significant difference between the SqueezeNet and the GoogLeNet. Figure 2b depicts the comparison of the Scc ICDF of both CNN architectures. The GoogLeNet is far better than the SqueezeNet because it assigns higher scores to the correct class. Moreover, the GoogLeNet has a lower slope in the central part of the graph. This means that Scc assigned have small variation. The SqueezeNet instead assigns Scc with higher variation.

We chose the GoogLeNet as the CNN architecture to port on the mobile phone because this architecture is the best approach in term of accuracy on ETHZ dataset (i.e. 73%). Moreover, it is second only to the SqueezeNet in terms of model size (about 40Mb vs 3Mb). Since the mobile memory can easily store 40Mb, we preferred the best accuracy. The SqueezeNet is slightly faster than the GoogLeNet due to its layers, but the speed of GoogLeNet is expected to be higher as soon as the mobile computational power increases.

We made a further analysis on the chosen CNN architecture: the Threshold evaluation. The purpose of this analysis is to establish a threshold on the score of the Top1 prediction in order to find the minimum score needed by the Top1 prediction to be considered correct.

The scheme in Figure 3 depicts the usage of the threshold. If the CNN assigns a score greater than the threshold to a class, it is confident with that prediction and therefore it can be directly shown to the user. If the CNN is not confident with the prediction, the list of Top5 predictions is shown to the user.

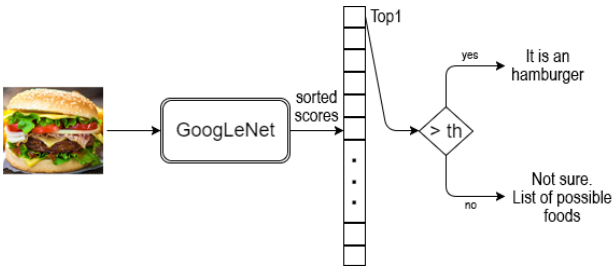


Fig. 3: Threshold usage

The graph in Figure 4 has been obtained by varying the threshold and by counting the number of False Positives (FP) and False Negatives (FN). The result of the test is classified as Positive if the Top1 prediction score is higher than the threshold, whereas it is classified as Negative if the Top1 prediction's score is lower than the threshold. We zoomed the graph in order to better show the cross-point which is the most important part of it. The lower is the cross-point on y-axis, the higher is the accuracy. On the y-axis, the lower is the crossing point of the two functions, the better is the CNN. If the crossing point of the FP and the FN function is 0, by setting the threshold at that point both zero FP and FN are obtained. Moreover, also the maximum accuracy is reached. Table 6 shows the results obtained by choosing different thresholds.

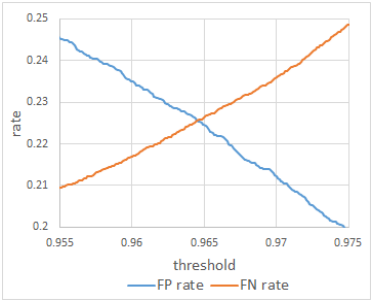


Fig. 4: Zoom on the cross-point of the threshold analysis

Threshold	Result	Scc ICDF (threshold)
0.7838	less than 10% of FN	65%
0.9785	minimum FP+FN	54%
0.9924	less than 10% of FP	49%

Table 6: Results with different thresholds

5 Mobile application development

Our proposal consists of an Android mobile application with two different modalities: on-line and offline.
In the online mode, the CNN is deployed on a server and therefore it is implemented in Caffe, which is imported in a Java Web Application. In this modality, the smartphone acts as a client and queries the CNN on the server.
In the off-line mode, the CNN is implemented relying on RenderScript. RenderScript is a framework for utilizing heterogeneous computing on Android phones. During execution, the runtime engine distributes the computation on available processing elements such as CPU cores and GPU. The used implementation of the GoogLeNet has been presented in [7]. In our work, we modified this CNN so that it can be used in Android phones. Figure 5 shows the architecture of the Android Application. The whole code of both the application and the Web Application is available on GitHub [2].

6 Conclusions and Future Work

In this work, a mobile application for food recognition has been designed, implemented, and tested. The application recognizes images representing food by predicting the dish. The application uses a Convolutional Neural Network which

[2] https://github.com/m1k3lin0/FoodRecognitionAndroidApp

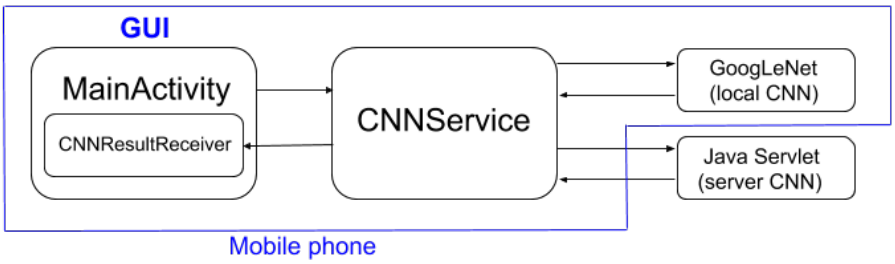


Fig. 5: Application architecture in blocks

has been deployed on a mobile phone after a training on two state-of-art datasets for food recognition.

In order to identify the most promising approach in terms of accuracy on the validation set, eight different CNN architectures have been trained and tested. Two scenarios have been identified: the SqueezeNet, which has a good level of accuracy and a model size extremely limited (i.e. 3Mb of model size and about 60% of accuracy), and the GoogLeNet, which has the best accuracy and a model size second only to the SqueezeNet (i.e. 40Mb of model size and about 70% of accuracy).

The mobile application has two versions: server-based version (i.e. CNN running on a Web Application) and local version (i.e. CNN developed by taking advantage of RenderScript framework). Hence, tests have been done in order to asses:

- The ability of the CNN to correctly predict the class of food represented in the image.
- The ability of the CNN to distinguish the correct prediction from the wrong prediction by establishing a threshold on the score assigned to the Top1 prediction.

The best scenario has been obtained by training and testing on ETHZ dataset. This is also the result that fits better with the application purpose because of the nature of the dataset (i.e. images with strong selfie style).
The GoogLeNet CNN is the best approach in terms of accuracy, but the SqueezeNet CNN is the best in terms of model size. The CNN used in both application modalities is the GoogLeNet.

References

1. Amato, G., Bolettieri, P., Monteiro de Lira, V., Muntean, C.I., Perego, R., Renso, C.: Social media image recognition for food trend analysis. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1333–1336. SIGIR '17, ACM, New York, NY, USA (2017). https://doi.org/doi:10.1145/3077136.3084142, http://doi.acm.org/10.1145/3077136.3084142

2. Bossard, L., Guillaumin, M., Van Gool, L.: Food-101 – Mining Discriminative Components with Random Forests, pp. 446–461. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10599-4_29, http://dx.doi.org/10.1007/978-3-319-10599-4_29
3. Chen, J., Ngo, C.w.: Deep-based ingredient recognition for cooking recipe retrieval. In: Proceedings of the 2016 ACM on Multimedia Conference. pp. 32–41. MM '16, ACM, New York, NY, USA (2016). https://doi.org/doi:10.1145/2964284.2964315, http://doi.acm.org/10.1145/2964284.2964315
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2016)
5. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. CoRR **abs/1602.07360** (2016), http://arxiv.org/abs/1602.07360
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012), http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
7. Motamedi, M., Fong, D., Ghiasi, S.: Fast and energy-efficient CNN inference on iot devices. CoRR **abs/1611.07151** (2016), http://arxiv.org/abs/1611.07151
8. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. CoRR **abs/1603.05279** (2016), http://arxiv.org/abs/1603.05279
9. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Li, F.: Imagenet large scale visual recognition challenge. CoRR **abs/1409.0575** (2014), http://arxiv.org/abs/1409.0575
10. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014), http://arxiv.org/abs/1409.1556
11. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2015)
12. Wang, X., Kumar, D., Thome, N., Cord, M., Precioso, F.: Recipe recognition with large multimodal food dataset. In: 2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW). pp. 1–6 (June 2015). https://doi.org/doi:10.1109/ICMEW.2015.7169757