Convolutional Neural Network for pixel-wise skyline detection

Darian Frajberg, Piero Fraternali, Rocio Nahime Torres

Politecnico di Milano, Piazza Leonardo da Vinci, 32, Milan, Italy
`{first.last}@polimi.it`

Abstract. Outdoor augmented reality applications are an emerging class of software systems that demand the fast identification of natural objects, such as plant species or mountain peaks, in low power mobile devices. Convolutional Neural Networks (CNN) have exhibited superior performance in a variety of computer vision tasks, but their training is a labor intensive task and their execution requires non negligible memory and CPU resources. This paper presents the results of training a CNN for the fast extraction of mountain skylines, which exhibits a good balance between accuracy (94,45% in best conditions and 86,87% in worst conditions) and runtime execution overhead (273 ms on a Nexus 6 mobile phone), and thus has been exploited for implementing a real-world augmented reality applications for mountain peak recognition running on low to mid-end mobile phones.

1 Introduction

Convolutional Neural Networks (CNNs) are a powerful tool for addressing hard object recognition tasks, and have achieved significant improvements outperforming previous computer vision techniques in many benchmarks. In particular, detection problems such as biomedical images analysis [3] and edges extraction [9] require to be solved with high precision at pixel level. An emerging field of application of CNNs is the implementation of *Augmented Reality (AR)* systems, in which the users are offered an interface that enriches the view of real objects with computer-generated information [5]. AR applications are normally implemented on portable, low power devices, such as smart glasses or even mobile phones. Examples are found in tourism (e.g., PeakLens[1]), astronomy (e.g, Star Chart[2]), games (e.g., PokemonGo[3]), etc. The main challenge of developing a computer vision component for an AR application for low power devices is the need of providing high recognition accuracy, real-time performance, with acceptable memory and battery consumption. These competing objectives require adequate training of the CNN, minimization of the model, and reduction of the overall application fingerprint. This paper presents the training and evaluation of a CNN for a pixel-wise mountain skyline detection task and reports

[1] http://www.peaklens.com
[2] http://www.sites.google.com/site/starchartuserguide
[3] http://www.pokemongo.com

the result of its usage in the development of PeakLens [4], an AR mobile app for mountain peaks identification. PeakLens relies on the alignment between a virtual panorama visible from the current user's location, computed from the GPS coordinates, from the compass orientation, and from a Digital Elevation Model (DEM) of the Earth, and the mountain skyline extracted from the camera view. Such alignment must be done precisely and in real time and thus requires very high accuracy of the extracted skyline and fast execution. For training and evaluating the skyline detection CNN, we executed a crowd-sourcing task and manually annotated 8.940 mountain images, fetched from Flickr and from over 2.000 publicly available touristic web-cams. Images in the data set are complex, diverse, and contain a variety of obstacles occluding the skyline horizon.

The focus of this paper is the description of the CNN model and of its training, and the evaluation of the resulting pixel-level classifier for mountain images taken in uncontrolled conditions. The training data set consists of positive and negative patches automatically sampled from the annotated mountain photo collection. For evaluation, we noted that the CNN accuracy obtained at patch level does not represent well the quality of the output for an entire image; thus, we defined metric functions that assess the quality of the mountain skylines extracted with the CNN at the whole image level, and computed it with the help of the manually annotated ground truth skylines. The contributions of the paper are the following:

- We recall the mountain skyline extraction problem, as defined in the relevant literature (e.g., [2]).
- We illustrate a mountain skyline extraction pipeline that exploits a CNN for evaluating the probability of pixels to belong to the skyline, and a post-processing step for extracting the actual skyline from pixel probabilities.
- We define whole image metrics for evaluating the quality of an extracted skyline with respect to the gold standard created with annotations.
- We evaluate the designed pipeline quantitatively, in terms of precision and execution overhead (time and memory). Precision is evaluated on two classes of images: without occlusions and with occlusions.
- We discuss the use of the realized component in a real world mobile app (www.peaklens.com), with very demanding accuracy, speed and memory constraints.

The paper is organized as follows: Section 2 briefly surveys the related work on pixel-wise feature and objected detection and on the specific problem of mountain skyline detection; Section 3 explains the proposed CNN models and the method to train and evaluate it; Section 4 presents the result of evaluating the accuracy and performance of the trained skyline extraction component and also discusses a real-world case study where the component is embedded in a mobile app for mountain peak detection; finally, Section 5 concludes and gives an outlook on the future work.

2 Related Work

Skyline extraction is a sub-problem of image-to-terrain alignment; early works, such as [1] and [2], tackled the problem by computing the alignment between the digital elevation model (DEM) and skylines extracted (offline at the server-side) from mountain images.

Heuristic methods. To extract skylines, [2] proposed an automatic approach exploiting an edge-based heuristics, whereas [1] applied sky segmentation techniques based on dynamic programming, which required manual support for challenging pictures. The authors of [1] also released a dataset that contains 203 images with ground truth information (including segmentation masks). Feature-based heuristic methods (e.g., based on edge detection) work well on images taken in good conditions, but do not address adequately bad weather and skyline occlusions. In these cases, a cloud, a high voltage cable, or a roof impact negatively the heuristic edge filter, e.g., a cloud edge would be treated as skyline and the mountain slope below would be considered as noise.

CNN methods. Skyline extraction problems can also be addressed with CNNs, which have exhibited superior performance in a variety of computer vision tasks, such as object recognition and semantic segmentation. In [8] the authors used the dataset of [1] to extract the skyline with a deconvolutional neural network for image segmentation; their approach treats an input image as a foreground-background segmentation problem and does not single out obstacles. Pixel-level CNN methods have been experimented successfully e.g., in biomedical images analysis. In [3] the authors proposed a novel binary pixel-based CNN for the detection of mitosis in breast cancer histology images. The network is trained with patches extracted from the images, classified as mitosis or non-mitosis based on the probability of the center pixel of being close to the centroid of a mitosis. Pixel-wise CNNs are also used for edges extraction problems. In [9] the authors take image patches as input and predict whether their central pixels belong to an edge.

Our skyline detection approach is inspired to [3] and [9] and also works at pixel-level. To treat images taken in uncontrolled conditions and cope with obstacles of different nature (people, bell towers, cables, etc), we consider an image as a map of patches and analyze the local context around each center pixel to predict whether it belongs to the skyline. Differently from [3] and [9] we specialize the CNN to mountain skyline detection; differently from [8], we train the network on a large data set of images (8.940) taken in uncontrolled conditions including samples with many different types of obstacles, and we evaluate the obtained precision quantitatively (94,45% in best conditions and 86,87% in worst conditions). Differently from all the mentioned works, we target the fast execution of the CNN on low power devices and report the performance of skyline extraction (273 ms per image on a Nexus 6 mobile phone).

3 Skyline extraction with CNN

We defined the CNN architecture presented in Table 1, which is an adaptation of the well known LeNet model [7]. The main difference is that we do not consider 28x28 gray-scaled inputs, but 29x29 RGB inputs. The output of our architecture considers 2 classes, which represent whether the center pixel of the input image is part of the skyline (1) or not (0). In the sequel, we will consider the probability of a pixel to be part of the skyline.

Layer	**Type**	**Input**	**Filter**	**Stride**	**Pad**	**Output**
Layer 1	Conv	29 x 29 x 3	6 x 6 x 3 x 20	1	0	24 x 24 x 20
Layer 2	Pool (max)	24 x 24 x 20	2 x 2	2	0	12 x 12 x 20
Layer 3	Conv	12 x 12 x 20	5 x 5 x 20 x 50	1	0	8 x 8 x 50
Layer 4	Pool (max)	8 x 8 x 50	2 x 2	2	0	4 x 4 x 50
Layer 5	Conv	4 x 4 x 50	4 x 4 x 50 x 500	1	0	1 x 1 x 500
Layer 6	Relu	1 x 1 x 500	Max(0,x)	1	0	1 x 1 x 500
Layer 7	Conv	1 x 1 x 500	1 x 1 x 500 x 2	1	0	1 x 1 x 2
Layer 8	Softmaxloss	1 x 1 x 2	-	1	0	1 x 1 x 2

Table 1. CNN architecture

To create the input dataset, we conducted an internal crowd-sourcing task and manually annotated the skyline of 8.940 mountain images fetched from Flickr and from over 2.000 publicly available touristic web-cams. Images in the data set are complex, diverse and contain a variety of obstacles occluding the skyline horizon[4]. The dataset images were split 65% for training, 25% for validation and 10% for test. The preparation of the training data set consisted in the extraction of positive and negative patches from the mountain photo collection. For such purpose, we applied for each image a soft Canny filter and computed the edge map to enhance the selection of candidate patches, labeling them as positive or negative based on their center pixel. When an edge point and an annotation point match, that pixel is considered to be positive; while if an edge point has no match, it is considered negative. Since non-skyline points are much more numerous than skyline points, we generated an unbalanced dataset by randomly extracting 100 positive and 200 negative patches from each image. The CNN model is trained using the Caffe framework [6] on a machine with an NVIDIA GeForce GTX 1080. It took 61 minutes to complete and the total number of learned parameters of the resulting model was 428.732. At execution time, the fully convolutional network is fed with whole images and returns a spatial map for each image, in which each pixel is assigned a probability of being positive.

Post-processing A post-processing step was executed over the skylines extracted with the CNN so as to select at most the N pixels per column (PPC) with the highest probability score. Such step consisted in the application of a

[4] A large sample is visible at https://goo.gl/g4lygB

small erosion and the removal of all the pixels in rows with positive probability scores lower than a predefined threshold (THR). In addition, we tested also the outcome of post-processing the CNN output with the multiplication between its content and a canny edge map extracted from the original image.

4 Evaluation

This section describes the evaluation performed to measure the accuracy of the skyline identification, the runtime performance, and a concrete usage experience.
Accuracy. The maximum accuracy achieved by the CNN model over the test dataset **at patch level** was 95,05%, obtained with a threshold value for positive probability of 0,4644. However, accuracy measured at patch level does not represent intuitively the quality of the output for an entire image. Therefore, we defined metric functions that assess image level quality by comparing the skyline extracted with the CNN with the one manually annotated in the ground truth. Then we evaluated such functions on the test dataset images. Let, $CNN(i,j)$ be a function that returns 1 if the image pixel at coordinates (i,j) belongs to the skyline extracted by the CNN (0 otherwise) and let $GT(i,j)$ be a function that returns 1 if the pixel (i,j) belongs to the ground truth skyline (0 otherwise). The following image-level metric functions have been used:

$$ASA = \sum_{j=1}^{cols} I_{GT \wedge CNN}(j) / \sum_{j=1}^{cols} I_{GT}(j) \tag{1}$$

$$ANSA = \sum_{j=1}^{cols} I_{\overline{GT} \wedge \overline{CNN}}(i,j) / (cols - \sum_{j=1}^{cols} I_{GT}(j)) \tag{2}$$

$$AA = \frac{1}{cols} \sum_{j=1}^{cols} I_{agree}(j) \tag{3}$$

where:
$I_{GT}(j) := 1\ if\ \exists i|\ GT(i,j) = 1;\ 0\ otherwise$
$I_{GT \wedge CNN}(j) := 1\ if\ \exists i|\ GT(i,j) = 1 \wedge\ CNN(i,j) = 1;\ 0\ otherwise$
$I_{\overline{GT} \wedge \overline{CNN}}(j) := 1\ if\ \forall i|\ GT(i,j) = 0 \wedge\ CNN(i,j) = 0;\ 0\ otherwise$
$I_{agree}(j)\ := 1\ if\ I_{GT \wedge CNN}(j) = 1 \vee I_{\overline{GT} \wedge \overline{CNN}}(j) = 1;\ 0\ otherwise$

ASA (Average Skyline Accuracy) measures the fraction of image columns that contain ground truth skyline pixels and in which at least one of the positive (i.e., above threshold) pixels extracted by the CNN matches one of the ground truth pixels; **ANSA (Average No Skyline Accuracy)** measures the fraction of columns that do not contain any ground truth skyline pixel (due to obstacles) and for which also the CNN output does not contain positive pixels; this metric evaluates the presence of false positives in images with an interrupted ground truth skyline; **AA (Average Accuracy)** measures the fraction of columns in which the ground truth and the CNN skyline agree, considering agreement when none contain pixels or otherwise at least one of the CNN pixels matches one of the ground truth pixels.

Figure 1 shows a mountain image with the ground truth annotation (left) and the value of the quality metrics calculated on the output produced by a CNN with post-processing that selects 1 PPC. On the right, pixels in green represent correctly predicted skyline pixels, while pixels in red represent incorrect ones.

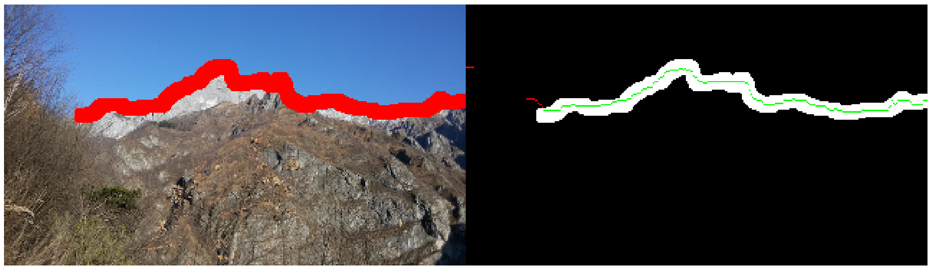

Fig. 1. Evaluation of image with interrupted skyline. Average Skyline Accuracy: 98%. Average No Skyline Accuracy: 73%. Average Accuracy: 94%.

To evaluate the quality loss due to occlusions that produce non continuous skylines, we performed two evaluation rounds. First, we assessed the 462 images of the test dataset (51,68%) with no interruptions. As shown in Table 2 Average Accuracy is 94,45% with 1 PPC (row 1) and 97,04% with 3 PPC (row 6). The threshold was set to 0 to maximize the chances of selecting at least one CNN pixel per column. The loss of accuracy is only due to pixels that the CNN positions at a different row w.r.t. the ground truth. In the second round, we considered the entire test dataset of 894 images, in which 8% of all the columns correspond to interrupted skyline. Results are reported in Table 3: the maximum Average Accuracy decreases to 86,87% for 1 PPC (row 7) and 89,36% for 3 PPC (row 12): occlusions that interrupt the skyline impact the accuracy by introducing false positives and false negatives. A threshold of 0,3921 proved the most suitable value to maximize the AA metric. Different post-processing methods were also evaluated, as shown in the other rows of Table 2 and 3: overall the use of the CNN without post-processing achieved the highest result with 1 PPC (Table 2, row 1: 94,45%; and 3 row 7: 86,87%), such values improve when the multiplication between the CNN output and Canny is used, achieving the best results with 3 PPC (Table 2, row 6: 97,04% and Table 3, row 12 89,36%); the multiplication between the extracted skyline and the edge map obtained with a Gaussian Blur followed by a Canny filter was always outperformed.
Runtime Performance. The execution of the CNN model in desktop PCs has negligible execution time per image. To evaluate the suitability for an AR application on low power mobile devices, where not only a high recognition accuracy is needed, but also a real-time performance, we assessed the execution time per image in smart-phones of different categories. To this end, we selected an input image of dimensions 321 x 241 pixels. While most smart-phones support taking pictures of larger size, after different experimental trials we observed that this

V	PPC	THR	Multiplication	ASA	ANSA	AA
1	1	0	No	94,45%	-	**94,45%**
2	1	0	(Blur + Canny)	92,69%	-	92,69%
3	1	0	Canny	93,73%	-	93,73%
4	3	0	No	95,92%	-	95,92%
5	3	0	(Blur + Canny)	96,72%	-	96,72%
6	3	0	Canny	97,04%	-	**97,04%**

Table 2. Evaluation on test dataset with only continuous skyline images

V	PPC	THR	Multiplication	ASA	ANSA	AA
7	1	0,3921	No	92,45%	20,14%	**86,87%**
8	1	0,3921	(Blur + Canny)	90,11%	28,77	85,31%
9	1	0,3921	Canny	91,55%	23,19%	86,21%
10	3	0,3921	No	94,25%	18,83%	88,41%
11	3	0,3921	(Blur + Canny)	93,97%	28,65%	88,83%
12	3	0,3921	Canny	95,07%	22,54%	**89,36%**

Table 3. Evaluation on complete test dataset

dimension has the best balance of accuracy, memory consumption (9.36MB on average), and execution time, on a broad spectrum of devices. The evaluation was performed by repeating skyline extraction on a test image 1.000 times in each device, taking as result the average execution time. As shown in Table 4,

Device	**Time(ms)**
MacBook Pro - 2,9GHz Intel Core i5 (2 cores) - 16GB	73
Nexus 6 - 2.65GHz Qualcomm Snapdragon 805 (4 cores) - 3GB	273
One Plus A0001 - 2.46 GHz Qualcomm Snapdragon 801 (4 cores) - 3GB	296
Nexus 5X - 1.82GHz Qualcomm Snapdragon 808 (2 cores) - 2GB	437
Moto G4 PLUS - 1.52GHz Qualcomm Snapdragon 617 (8 cores) - 2GB	472
Asus Z00D - 1.6 GHz Intel Atom z2560 (2 cores) - 2GB	1128
Galaxy Nexus - 1.2GHz TI OMAP 4460 (2 cores) - 693 MB	1775

Table 4. Time required to execute the skyline extraction

the execution time in low power mobile devices is much higher than in a PC, where skyline extraction can be performed at a frequency of 13 images per second: the best smart-phone of the test could process around 3 images per second, whereas computation took something less than 2 seconds in the devices with lowest hardware. Processing images at the rates shown in Table 4 is compatible with the usability requirements of a real time AR application. Image processing is done in background with respect to the user interface; if the camera view movements are not too sudden, as one expects in a mountain peak recognition use case, the skyline extraction and the subsequent DEM alignment step could

be done at a frequency lower that the 15 frame per second normally considered viable for video play; the price to pay is some jitter in the camera view, when the skyline extraction and DEM alignment execute and require an update of the peak positions in the camera view. However, this limitation on low power mobile phones did not seem to impact users too much, as discussed next.
Usage experience. The skyline extraction CNN described in the paper is embedded in the PeakLens AR mobile app, which provides real time mountain peak identification by processing camera frames at the maximum allowed speed and overlaying onto them the icons of the visible mountain peaks. Peaks are fetched by an annotated DEM, queried on-line, when Internet connectivity is available, or off-line on board of the mobile phone, where it is stored in a compressed format. The initial peak positioning is done using only the DEM and the GPS and compass sensors: the virtual panorama in view is estimated and peaks are projected onto the camera frame based on where they are in the virtual panorama. Obviously, such method is extremely prone to the frequently occurring errors in the DEM, GPS and compass. Here is where the skyline extraction component is exploited, by updating the peak positions based on the registration of the camera view skyline extracted by the CNN and the skyline of the virtual panorama. Thanks to such registration, PeakLens is able to automatically correct substantial errors in the DEM, GPS position and compass, in real time.

5 Conclusions and future work

In this paper we have discussed a CNN model for mountain extraction skyline, trained with a large set of annotated images taken in uncontrolled conditions, and capable of supporting an AR mountain peak recognition app also on low-end mobile phones. Future work will concentrate on the optimization of the CNN model, to make its execution faster on phones with less on 1GB RAM and support usage even without the compass, which requires the very fast alignment of the camera view with a 360 degree virtual panorama.

References

1. Baatz, G., Saurer, O., Köser, K., Pollefeys, M.: Large scale visual geo-localization of images in mountainous terrain. Computer Vision–ECCV 2012 pp. 517–530 (2012)
2. Baboud, L., Čadík, M., Eisemann, E., Seidel, H.P.: Automatic photo-to-terrain alignment for the annotation of mountain pictures. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. pp. 41–48. IEEE (2011)
3. Cireşan, D.C., Giusti, A., Gambardella, L.M., Schmidhuber, J.: Mitosis detection in breast cancer histology images with deep neural networks. In: International Conference on Medical Image Computing and Computer-assisted Intervention. pp. 411–418. Springer (2013)
4. Fedorov, R., Frajberg, D., Fraternali, P.: A framework for outdoor mobile augmented reality and its application to mountain peak detection. In: International Conference on Augmented Reality, Virtual Reality and Computer Graphics. pp. 281–301. Springer (2016)

5. Jain, P., Manweiler, J., Roy Choudhury, R.: Overlay: Practical mobile augmented reality. In: Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services. pp. 331–344. ACM (2015)
6. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia. pp. 675–678. ACM (2014)
7. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
8. Porzi, L., Rota Bulò, S., Ricci, E.: A deeply-supervised deconvolutional network for horizon line detection. In: Proceedings of the 2016 ACM on Multimedia Conference. pp. 137–141. ACM (2016)
9. Wang, R.: Edge detection using convolutional neural network. In: International Symposium on Neural Networks. pp. 12–20. Springer (2016)