KCNN: Extremely-Efficient Hardware Keypoint Detection with a Compact Convolutional Neural Network

Paolo Di Febbo[1], Carlo Dal Mutto[1], Kinh Tieu[1], Stefano Mattoccia[2]
[1]Aquifi Inc. [2]University of Bologna
{paolo,cdm,ktieu}@aquifi.com, stefano.mattoccia@unibo.it

Abstract

Keypoint detection algorithms are typically based on handcrafted combinations of derivative operations implemented with standard image filtering approaches. The early layers of Convolutional Neural Networks (CNNs) for image classification, whose implementation is nowadays often available within optimized hardware units, are characterized by a similar architecture. Therefore, the exploration of CNNs for keypoint detection is a promising avenue to obtain a low-latency implementation, also enabling to effectively move the computational cost of the detection to dedicated Neural Network processing units. This paper proposes a methodology for effective keypoint detection by means of an efficient CNN characterized by a compact three-layer architecture. A novel training procedure is proposed for learning values of the network parameters which allow for an approximation of the response of handcrafted detectors, showing that the proposed architecture is able to obtain results comparable with the state of the art. The capability of emulating different detectors allows to deploy a variety of algorithms to dedicated hardware by simply retraining the network. A sensor-based FPGA implementation of the introduced CNN architecture is presented, allowing latency smaller than $1[ms]$.

1. Introduction

Keypoint detectors are fundamental components in many computer vision systems. Applications of keypoint detection include tracking and 3D reconstruction, which often have extremely low latency and power efficiency requirements, such as in the case of autonomous driving (latency) and AR/VR pose estimation (latency and power consumption). The majority of state of the art keypoint detectors [3, 12, 5, 14, 13] are based on combinations of derivative operations, such as determinant of the Hessian [3] or difference of Gaussians [12], and their implementations are based on conventional image filtering and processing approaches.

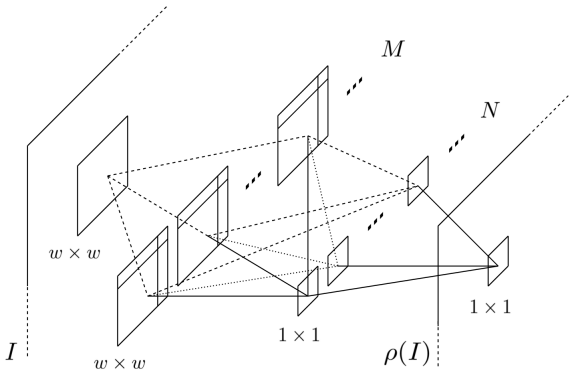


Figure 1: Detector architecture, seen as a Fully Convolutional Neural Network. I is the input image, and $\rho(I)$ is the keypoint response output. Each window corresponds to a convolution operation. The filters on the first layer are separable. Equation 1 describes the network in detail.

Similarly to keypoint detectors, the early layers of Convolutional Neural Networks (CNNs) are also characterized by combinations of filtering operations, hinting that keypoint detectors could be implemented as CNNs. This paper focuses on state of the art keypoint detectors, showing that it is possible to build a CNN-based system that matches or exceeds their performance. In particular, a fundamental contribution of this paper is the design of a CNN architecture and training methodology that is able to obtain results comparable with state of the art of keypoint detectors, namely KAZE [3], while surpassing the seminal SIFT detector [12] and modern machine-learning based algorithms [28, 26].

Keypoint detection implemented as a CNN is not only interesting from a theoretical point of view, but also leads to practical benefits. In particular, a design of this kind enables the inclusion of keypoint detection within a more complex system based on CNNs as well, allowing for end-to-end system training [28]. Another fundamental advantage of CNN implementation with respect to handcrafted algorithms is that CNN development is constantly evolv-

ing in both speed and power consumption. In particular, CNNs are often implemented on GPGPUs, which are characterized by ever-increasing speed and power efficiency, on advanced hardware solutions that have been developed for performance (*e.g.*, Google TPU) and power efficiency (*e.g.*, Movidius Fathom) and they can also be implemented on FPGAs (Field Programmable Gate Arrays) or silicon-based architectures. Therefore, a CNN implementation of keypoint detection implicitly captures the benefits of this independent implementation improvement. Clearly, the downside of a pure hardware implementation would be the lack of reconfigurability. In fact, the implementation of a new algorithm is in general extremely time consuming and it is gated by the limited resources. The proposed approach overcomes this constraint by the introduction of a small network architecture that is able to emulate multiple keypoint detection algorithms, such as KAZE [3] and SIFT [12], and which is claimed to be possibly suited also for algorithms yet to be proposed. The emulation of a current or novel algorithm can be simply regarded as a CNN parameters update. The complexity of the algorithm that can be implemented in this way is directly connected to the capacity of the network.

For speed and power efficiency, which are particularly important in the case of embedded systems, it is desired to consider CNN architectures characterized by a small footprint, especially in the case of FPGA implementations. In this case, keypoints can be computed in a streaming-based fashion that does not require to store the entire content of the image, intercepting the sensor data stream at readout. The considered CNN is therefore designed as a compact three layers architecture with separable convolutional kernels and quantized filter weights resulting in less than $1[KB]$ of total memory utilization for the parameters.

In order to obtain a CNN that is able to approximate the output of any specific keypoint detector, it is important to devise an effective training methodology. In fact, in this case, training data can be obtained by simply running a keypoint detector on a set of images, so the amount of labeled training data is potentially unlimited. The fundamental issue in this problem regards instead the effective exploitation of this training data. This paper introduces a training scheme based on hard negative mining that adaptively selects the examples to be considered during training, in order to effectively learn the values of the convolution parameters.

Experimental results using standard metrics [17] show that the proposed network architecture coupled with the training methodology is able to effectively learn to produce the keypoint responses of state of the art algorithms, *i.e.*, KAZE [3] and of the well-known SIFT [12], outperforming other state of the art learning-based approaches [26, 28]. Power efficiency measurements and latency performance results of our streaming-based FPGA implementation of the network empirically prove the potential of having a dedicated CNN hardware implementation for keypoint detection, paving the way to mainstream applications that are particularly demanding.

2. Related Work

Detecting keypoints is a well-known problem and a great variety of approaches to tackle it have been proposed in the computer vision literature.

Many handcrafted keypoint detectors have been developed since Lowe's SIFT [12] in 1999, which uses difference of Gaussians as its response operator. MSER [15] introduced the unique concept of extremal regions, while [16] proposed an affine invariant detector. SURF [5] defined a fast approximation of the determinant of the Hessian response by using integral images. The determinant of the Hessian can still be considered as one of the best operators for robust keypoint detection [11], and it has been used by other algorithms such as recently by KAZE [3]. In addition to this operator, KAZE adds the novel concept of using a non-linear diffusion scale space instead of the more traditional Gaussian pyramid, making it the *de facto* current state of the art among handcrafted keypoint detectors. More recently, SIFER [14] and D-SIFER [13] proposed an advanced Cosine Modulated Gaussian filter instead of traditional derivative-based ones, with promising results.

Besides handcrafted algorithms, machine learning methods has also been heavily exploited in recent years. One of the first attempts in using machine learning techniques to speed up conventional detectors is FAST [20]. ORB [21] extended the basic concept of FAST and enhanced its capabilities, while extending it to the scale space. Some approaches focus on learning edges rather than keypoints [7]. Recently, it has been shown that instead of learning a detector from scratch, it is possible to enhance detectors by learning a score for evaluating matchability of keypoints [10].

Closer in spirit to our approach, the method of [23] shows that it is possible to emulate handcrafted detectors by learning them. This approach leads to interesting results, although it uses a WaldBoost classifier instead of CNNs. Another approach tunes for specific tasks such as detecting keypoints from man-made structures [24]. Other machine learning approaches include using Genetic Programming [25], and learning linear filters [19] where the lack of non-linearity makes it more suited for very specific tasks. [4] uses deep networks for keypoint detection with a hard negative mining training approach which is similar to ours, but applied in a different fashion and focusing on a custom dataset of aerial images. The state of the art for machine learning based solutions is represented by TILDE [26] and LIFT [28]. TILDE introduces a temporally invariant detector with an interesting loss function to enforce the response shape, although its performance was not compared to KAZE. LIFT defines a complete end-to-end solution for

detection and description of features based on SIFT. While very interesting from a theoretical point of view, the architecture of TILDE and LIFT cannot be easily implemented in streaming-fashion on a compact FPGA architecture due to their large computational and memory footprint.

3. Network Architecture

A novel, compact CNN architecture is defined for effectively performing keypoint detection with minimum layout complexity. The considered CNN takes as input a single channel image I of arbitrary size, and produces as output the relative keypoint response function $\rho(I)$ which has the same size as the input image. Per pixel, the response function gives a score which describes how likely that pixel is supposed to be a keypoint, according to the learned detector.

The network acts as an end-to-end image-to-response function, without requiring any pre-processing on the input image. A single instance of the network computes a single response function, meaning that the CNN needs to be instantiated as many times as the number of needed scales in order to compute the response for the whole scale space. In order to perform the actual keypoint detection, a simple non-maxima suppression algorithm can be run on the response maps after these are generated.

The architecture of the proposed CNN, represented in Figure 1, is constituted by three layers. The first layer of the network is made of M convolutional filters of size $w \times w$, generated by corresponding separable filters. Separable filters reduce the number of actual parameters for a single filter from w^2 to $2w$, making the network more compact. The second layer is made of N 1×1 convolutional filters, which perform N different linear combinations of the outputs produced by the first layer. Considered together, these first two layers actually allow for an approximation of standard convolutional filters, as proved by [22]. The last layer linearly combines the N outputs to produce the final response output. In between the different layers a ReLU [9] activation function is added to give the network the capability of approximating non-linearities.

Formally, the network can be described by the following function:

$$\rho(I) = \sum_{i}^{N} a_i \, \phi \left(\sum_{j}^{M} c_{ij} \, \phi \left(\boldsymbol{e}_j \boldsymbol{f}_j^T * I + g_j \right) + d_i \right) + b \quad (1)$$

where $\boldsymbol{e}_j$ and $\boldsymbol{f}_j$ are the separable convolutional kernels of the j-th filter, ϕ is the ReLU activation, and $\boldsymbol{a}$, b, C, $\boldsymbol{d}$, E, F, $\boldsymbol{g}$ are the parameters to be learned. The hyper-parameters M, N and w control the capacity of the network and its relative approximation capabilities. The function above can be interpreted as a non-linear regressor by replacing the input image I with a $w \times w$ patch $\boldsymbol{p}$ reshaped as a vector, the convolution operation with a dot product, $\boldsymbol{e}_j \boldsymbol{f}_j^T$ with its relative vector-shaped representation and the output $\rho(I)$ with a single scalar response output $\rho(\boldsymbol{p})$ for the given input patch. Considering this latter interpretation, the training can be performed on single patches of size $w \times w$ extracted from the images, and the resulting regressor can then be generalized to full images by just "sliding" it as a convolution.

4. Training Approach

Given a specific keypoint detector (*e.g.*, KAZE [3] or SIFT [12]), it is possible to naïvely regress its response output *as-is*. While this solution could possibly work, the different operators used for defining the response function used by different algorithms (*e.g.*, determinant of the Hessian, difference of Gaussians, *etc.*) lead to relative response domain inconsistency, introducing a further layer of complexity to be learned. Moreover, learning-based algorithms might rely only on the higher level definition of *good keypoints*, eventually described as position and scale (*e.g.*, learning features from SfM [10, 28], or manually labeled keypoints).

This leads to the conclusion that such naïve approaches are not ideal when different detectors need to be handled. Instead, regardless the detector it is always possible to define and synthesize a target response function r, defined for each pixel $\boldsymbol{p}$ within the image, which relies only on the knowledge of a set of detected keypoints $(\boldsymbol{k}, \pi) \in \mathcal{K}$, where $\boldsymbol{k}$ is the location of the keypoint and π its normalized response value. This allows to decoupling the proposed framework from specific image-based responses, leading to increased flexibility in choosing a variety of detectors that can be approximated. We define the response function r to be regressed as:

$$r(\boldsymbol{p}; \mathcal{K}) = \max_{(\boldsymbol{k},\pi) \in \mathcal{K}} A\pi \exp\left(-\frac{\|\boldsymbol{p} - \boldsymbol{k}\|^2}{2\sigma^2} \right), \quad (2)$$

where A is a defined amplitude coefficient and π models the strength of the detected keypoints. The choice of using the operator $\max$ for blending two keypoints aims at capturing the behavior of handcrafted algorithms: when two detected keypoints are very close, their original response is a wider blob of almost uniform intensity. A visual example of this generated response function is provided in Figure 2c.

Generating ground truth data for training can be done by simply evaluating the response r to be regressed on generic images. Given this ability of generating a possibly unlimited amount of labels, and that the size of the training set directly affects the training complexity, a strategy needs to be devised in order to select a small, well-distributed set of labels which guarantees that the overall function is represented. In order to effectively sample this domain, a novel iterative reinforcement method is proposed based on the following steps.

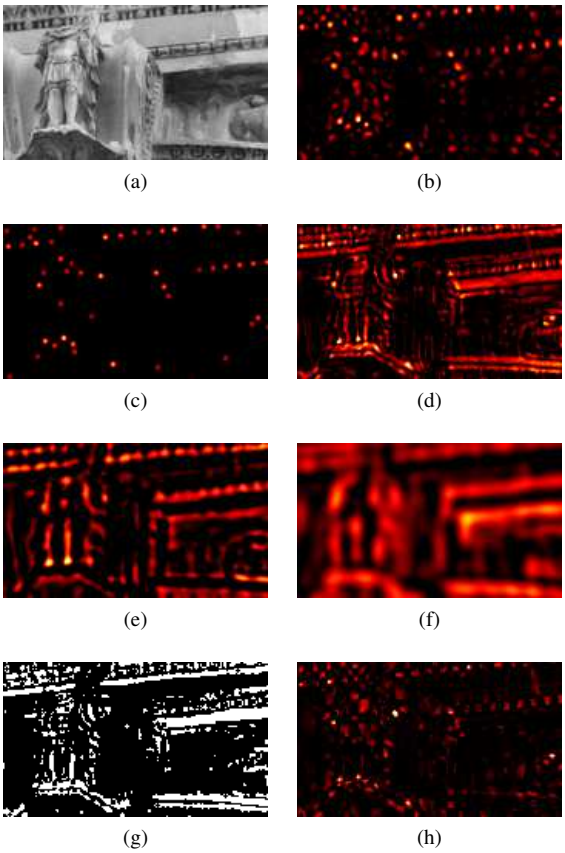


Figure 2: (a) Input image, (b) KAZE response, (c) generated response r, (d) response inference $\hat{r}$ from the network after initialization step. (e) and (f) are responses from respectively the state of the art LIFT [28] and TILDE [26] detectors, which do not use the proposed training set reinforcement process. (g) thresholded absolute difference $C(I;\cdot,\cdot)$ between (d) and (c) as described in Equation 3, (h) response inference $\hat{r}$ from the network after reinforcement. All responses are from the same scale, *i.e.*, the highest one. Best viewed on monitor.

(a) Define a set of images for training A set of images $\mathcal{I}$ is defined and their response r is computed per patch/pixel.

(b) Initialize training set The training set is initialized by sampling r uniformly in the response domain. For each image in $\mathcal{I}$, an histogram is built among the computed r values. Then, values are randomly selected within each bucket such that the same amount is picked per bucket.

(c) Perform the regression The training set above is used to train the network, obtaining the regressed response function $\hat{r}$. Figure 2d shows an example of the inferred response after the initialization step. Observe that the sampled information is not enough actually to represent the overall response function, as some specific components (*e.g.*, edges) are missing from the initial training set. Edges are false positives since by definition do not include useful unambiguous points to track. Notice that a similar behavior is noticeable in state of the art CNN-based detectors such as LIFT [28] and TILDE [26] as shown in Figures 2e and 2f.

(d) Reinforce the training set This step picks these under-represented components of r and adds more of these samples to the training set, enabling a better representation of the overall function. This is performed by thresholding the absolute difference of the inferred function and the actual function. Specifically, the following binary function is used to define whether a pixel (x, y) in the image I is a good candidate for being part of the reinforcement set or not:

$$C(I;x,y) = |T(r(I;x,y)) - T(\hat{r}(I;x,y))|, \qquad (3)$$

where T is a thresholding function defined as

$$T(f(I;x,y)) = \begin{cases} 1 & \text{if } f(I;x,y) \geq \max\limits_{(u,v)\in\Lambda_I} \dfrac{f(I;u,v)}{\theta} \\ 0 & \text{otherwise} \end{cases}, \qquad (4)$$

where Λ_I is the image matrix and the parameter θ controls the strength of the thresholding function. An example is shown in Figure 2g. Applying a threshold before the absolute difference instead of vice versa helps to enforce mostly points which are actually considered during the eventual keypoint detection stage, as thresholding is usually applied to the response before performing non-maxima suppression.

The points for which $C(I;x,y) = 1$ are added to the initial training set. Since their number is variable, random sampling can be performed to get a fixed amount proportional to the initial number of samples. Eventually, a number of randomly picked samples from the overall set of computed r values is added to the training set to tackle the classical reinforcement problem of balancing between exploration and exploitation[1].

(e) Iterate again from Step (c) Once the training set has been updated, a new training session is performed on the same network. Notice that the complexity of the network is not changing in any of the iterations, as it would otherwise happen in traditional boosting strategies with cascading classifiers. Instead, only the distribution of the training set is changed in order to better represent samples that are harder to learn.

[1]Although the connection between the proposed approach and reinforcement learning is intuitive, its formal analysis is beyond the scope of this paper.

This procedure can be iterated until satisfactory results are obtained (*i.e.*, convergence is reached or maximum number of iterations is met). Empirical results on the considered training data set (*i.e.*, Roman Forum [27]) show that a very small number of iterations is necessary to obtain convergence. In particular, the quantitative results reported in Section 6 are characterized by two iterations. An example of the final inferred result $\hat{r}$ is reported in Figure 2h.

5. Quantization

The capability of deploying this neural network on hardware for streaming-based real-time, low-power keypoint detection is one of the important goals which lead this design to be compact and efficient. Other than the size of the network itself, another crucial step for enabling an efficient hardware implementation is the use of fixed point arithmetic instead of the more standard floating point, introducing quantization of the network parameters.

The proposed approach is derived from the one proposed by [8], which however mostly focuses on fixed point computation for training networks. Instead, our solution only concerns the inference time rather than the training one, so their interesting clipping-or-rounding approach is here ported to the inference itself while the training stage is still performed with floating point values. The $\langle \texttt{IL}, \texttt{FL} \rangle$ notation is going to be used for describing a fixed point integer with IL integer bits and FL fractional bits.

Interpreting a generic neural network function as a series of dot products, the main problem when it comes to fixed point computation is that the result of a dot product requires a bit width which is bigger than the input one. When the output from a dot product is going to feed the subsequent dot product, and so forth and so on, the width of the result theoretically keeps growing making the computation too expensive. Specifically, for a dot product of n numbers of width $\langle \texttt{IL}, \texttt{FL} \rangle$ the relative result width is $\langle \log_2 n + 2\texttt{IL}, 2\texttt{FL} \rangle$. It is then necessary to define a way to crop this result width back to $\langle \texttt{IL}, \texttt{FL} \rangle$, such that all the operators within the network can be consistent to the same input size.

Inspired by [8] we define the following $\mathrm{Convert}$ function for this task. It is performed on the output of a generic dot product:

$$\mathrm{Convert}(x, \langle \texttt{IL}, \texttt{FL} \rangle) = \begin{cases} -2^{\texttt{IL}-1} & \text{if } x \leq -2^{\texttt{IL}-1} \\ 2^{\texttt{IL}-1} - 2^{-\texttt{FL}} & \text{if } x \geq 2^{\texttt{IL}-1} - 2^{-\texttt{FL}} \\ \lfloor x \rfloor & \text{otherwise} \end{cases} \quad (5)$$

Where $\lfloor x \rfloor$ is defined as the largest multiple of $2^{-\texttt{FL}}$ less than or equal to x. This function clips the input value x to the maximum or minimum value representable by $\langle \texttt{IL}, \texttt{FL} \rangle$ when it saturates, or crops the fixed point precision to the desired length otherwise.

The novelty of this approach is that, when it comes to converting the network's learned parameters from floating to fixed point, it only constraints the total bit width (*i.e.*, $\texttt{IL} + \texttt{FL}$) to be either 8 or 16 bit, while dynamically adapting the specific IL and FL widths according to the minimum needed IL size for each layer, thus maximizing the FL precision.

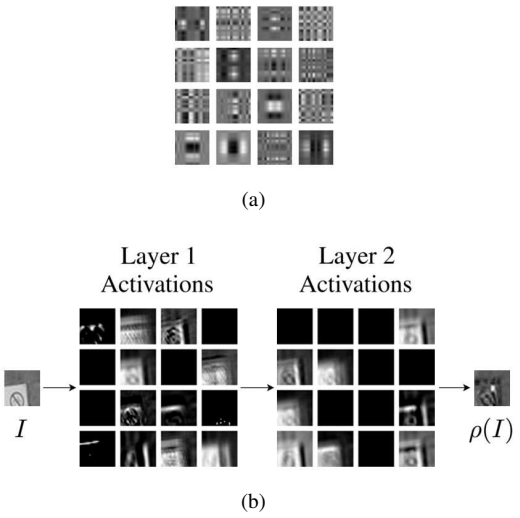
6. Experimental Results

For generating the training set, similarly to LIFT [28], we use images from the Roman Forum data set from [27]. Regarding the handcrafted detectors to be approximated, we consider KAZE [3] as the state of the art and SIFT to demonstrate that the network can be effectively trained using different algorithms as baseline. Hyper-parameters are set to $N = 16$, $M = 16$ and $w = 15$, leading to a total of 785 learned parameters including biases. With an 8-bit quantization, this means that the total size of the network is actually only 785 bytes. The choice of these specific values has been made in order to make the network compact enough to fit a cheap, small SoC FPGA solution such as the Xilinx Zynq 7020 or the Xilinx Artix 7 200T, showing that the network can provide good approximation performance even with a very low power, small hardware solution. Within training l_2 loss is used and batch size is set to 1000. In the initialization step, the number of samples selected per each of the 2000 considered images is 200. In the reinforcement step, 200 additional samples per image are added from $C(I; \cdot, \cdot)$ along with 200 more random samples per image as described in Section 4. The learned filters and an example activation of the network using KAZE as baseline are shown in Figure 3. Training time on a single GPU such as the NVIDIA GTX 1080 can be as fast as one hour, given the very limited amount of weights.

6.1. Quantitative Results

The performance of the detector is evaluated by training it independently with KAZE and SIFT, and comparing these two outputs against the handcrafted KAZE [3], SIFT [12], SURF [5] and the state of the art machine-learning based TILDE [26] and LIFT [28] (in their provided trained configuration) under the standard repeatability rate metric [17]. The repeatability rate measures the quality of a keypoint detector by considering a pair of images, which are related by a known homography transformation, and counting the number of keypoints that are repeatedly detected.

The standard Oxford image data set [17] has been chosen for this evaluation. This data set challenges the detectors on different conditions which include increasing image blur, viewpoint angle change, different zoom and rotations, lighting condition changes, and JPEG compression. All the con-



(a)

(b)

Figure 3: (a) learned filters from the first layer when trained with KAZE keypoints, (b) example activations for a typical corner.

sidered detectors have been configured such that the number of keypoints provided by each detector in the same image is as similar as possible.

Results are shown in Figure 4. The network successfully approximates state of the art KAZE when trained with the same algorithm. When trained with SIFT, the network generally performs better than the original algorithm, because the response r generated from the SIFT keypoints is more selective and filtered compared to the original simpler difference of Gaussians response. SURF, which detector is an approximation of the determinant of the Hessian operator, is one of the the closest alternative methods to our KAZE regressor. LIFT and TILDE use SIFT keypoints as baseline.

A note is necessary for Figures 4e and 4f: similarly to all the other cases in this dataset, zoom changes are applied increasingly along the x axis. Nevertheless, rotation changes are instead applied in a non-continuous scatter fashion [17]. This explains the scatter nature of these two plots, since for example image 5 happens to be rotated very similarly to image 2, although with a different zoom level.

6.2. Qualitative Results

An example of the detected keypoints in the Viewpoint (Graffiti) and Light scenes of the dataset is shown in Figure 5. The proposed KCNN detector is compared against the original KAZE, TILDE [26] and LIFT [28]. As shown in Figures 2e and 2f, the conventional training procedures of TILDE and LIFT mislead the detectors in positively considering edges as interesting keypoints. This directly reflects into the final output of these algorithms, where feature-wise identical points along high constrast edges are detected as shown in Figures 5c and 5d. Our approach is resilient to this type of false positives. Notice that metrics that purely consider detected keypoints in their evaluation (without description and matching), such as DTU [1] or even the considered Oxford [17] might wrongly assign positive scores to some of these false positives, as chances are high that random detected keypoints along one edge will match other random detected keypoints on the same edge in the second considered frame.

Algorithm	Device	Latency	Power
KAZE	Intel I7-5930K	58 ms	140 W
KCNN	Intel I7-5930K	289 ms	140 W
	NVIDIA GTX 1080	12 ms	180 W
	Xilinx Zynq 7020	1 ms	1.6 W
	Xilinx Artix 7 200T		

Table 1: Performance evaluation, considering the computation of a single-scale 1280×800 response.

6.3. Performance

The proposed CNN architecture has been deployed on CPU, GPU and FPGA, and its performance evaluated accordingly. In the first two, the Tensorflow [2] implementation of the network has been used, exploiting CUDA [18] acceleration in the GPU case. In the FPGA case, a custom hardware implementation has been designed and deployed to take advantage of dedicated silicon logic. Results are shown in Table 1, and compared against the standard KAZE CPU implementation from OpenCV [6]. It is possible to notice that deploying the proposed solution on a CPU is not recommended, as it runs slower than its handcrafted archetype on the same platform. Nevertheless it is clear that, as introduced earlier, the CNN solution is really effective on dedicated logic which include for instance standard GPUs and FPGAs. In case of GPUs, we achieved a speed-up of almost 5x compared to the CPU by just running the Tensorflow CUDA implementation *as-is*, without additional optimization and with a power consumption which is close to the one of the considered CPU. But an even more interesting result is achieved through custom gates in our FPGA implementation, which has 12x less latency and 100x smaller power consumption compared to the considered GPU. These results prove that dedicated silicon logic for CNNs can provide much less latency and very high power efficiency compared to more general purpose GPUs by orders of magnitude. Having a keypoint detector implemented on hardware can be crucial for guaranteeing real-time capabilities and power efficiency, while using a neural network decouples the hardware logic from the specific detector to be approximated allowing it to be reusable and deployable to dedicated CNN hardware.

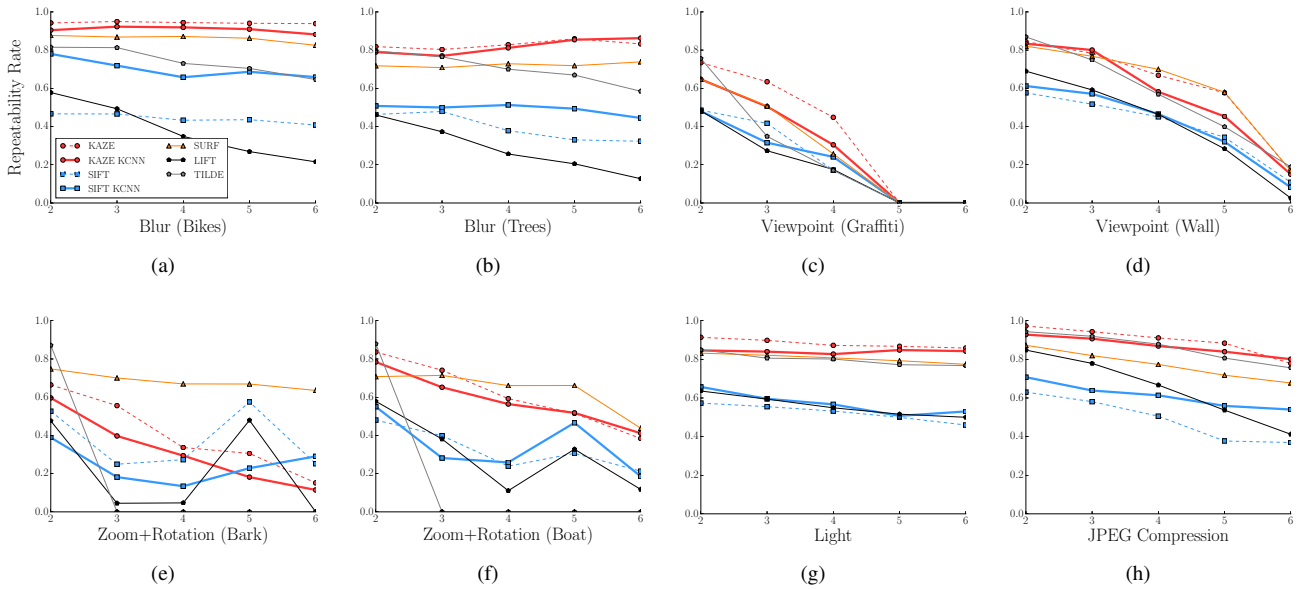
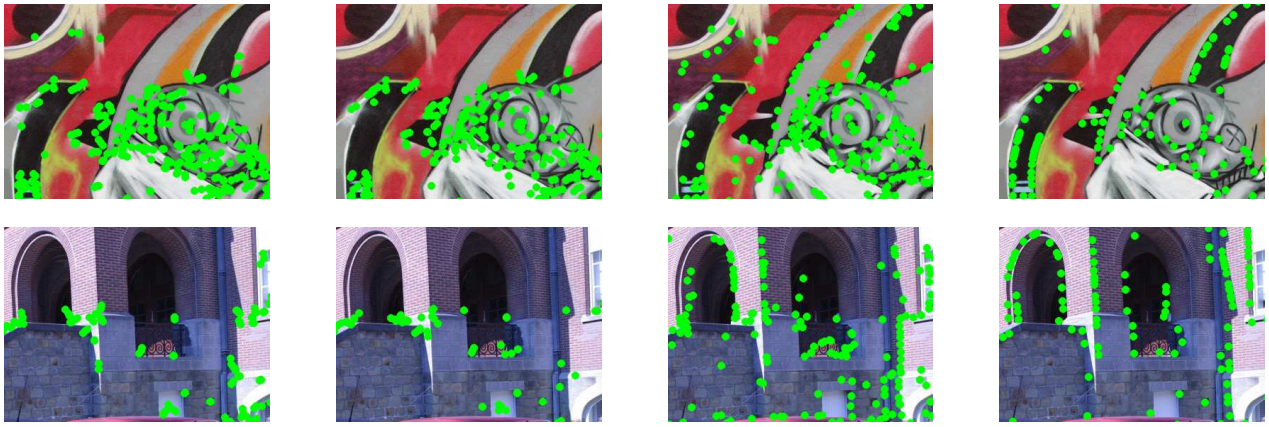


Figure 4: Quantitative results showing the Repeatability Rate on the standard Oxford data set [17]. (a) and (b) are challenging the detectors against increasing (x-axis) blur of the images while (c) and (d) are increasingly changing the viewpoint angle, (e) and (f) apply various scale transformations, (g) reduces the exposure and (h) the JPEG compression quality. The trained detector KAZE KCNN successfully approximates the state of the art KAZE and surpasses TILDE and LIFT, while being faster and more efficient.

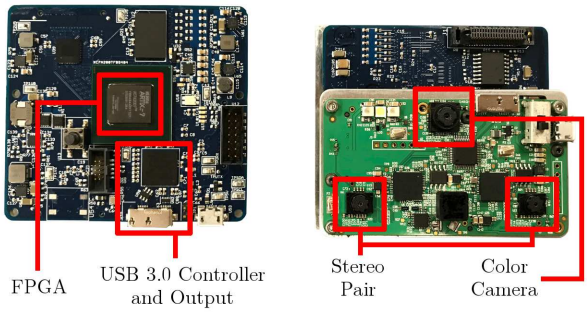


(a) KAZE (b) KAZE KCNN (c) LIFT (d) TILDE

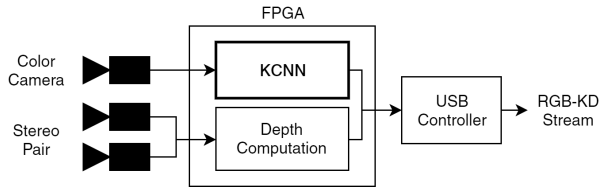
Figure 5: Visual comparison of detected keypoints from different considered algorithms. (a) original handcrafted KAZE algorithm, (b) learned KAZE from KCNN, (c) and (d) state of the art CNN-based detection algorithms TILDE [26] and LIFT [28]. Notice how the novel supervised reinforcement training of KCNN allows for a proper approximation of the learned algorithm, avoiding false positives along the edges. Best viewed on monitor.

7. Hardware Implementation

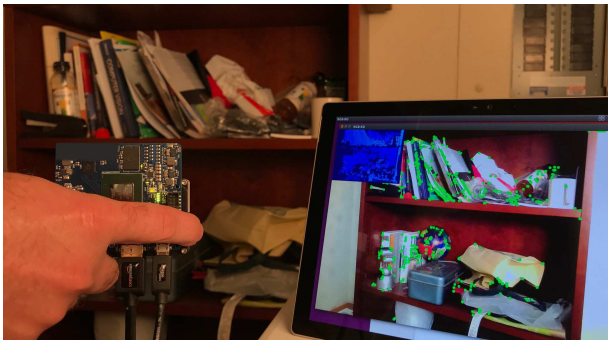
The KCNN detector has been deployed on a custom sensor board and fully integrated with a RGB-D sensor. Figure 6 shows a high level overview of the system. Three camera sensors are mounted on the board and connected to a Xilinx Artix 7 200T FPGA through independent MIPI lanes. Two of the sensors are used for active stereo depth computation with VGA resolution at $60[Hz]$ (not detailed in this paper), whereas the top one is used for capturing RGB data and performing keypoint detection through KCNN. The RGB



(a) Back of the board (b) Front of the board



(c) Architecture diagram



(d) RGB-KD module framing a stationary scene with live visualization on a monitor. Depth map is in the upper left corner. Keypoints are overimposed in green on the color stream. Best viewed on monitor.

Figure 6: Compact hardware system implementing KCNN along with a RGB-D sensor, *i.e.*, a *RGB-KD* device.

Resource	**Amount**	**% on Zynq 7020 SoC**	**% on Artix 7 200T FPGA**
LUT	21.120	40%	16%
FF	90.587	85%	34%
BRAM	4	3%	1%
DSP	196	89%	26%

Table 2: Hardware resources utilization for KCNN on Xilinx FPGA (successfully implemented in both Xilinx Zynq 7020 SoC and Xilinx Artix 7 200T FPGA).

image has resolution 1280×800 and has a framerate of $60[Hz]$ which is synchronized with the depth frames.

The FPGA implementation of KCNN computes the response map $\hat{r}$ *on-the-fly* directly from the camera sensor's real time pixel stream, buffering only a small number of frame lines (7), which are required by the neural network, introducing a latency smaller than $1[ms]$. Still at the sensor stream level, the response output is thresholded and non-maxima suppression is performed to obtain a binary map of the detected keypoints. The generated binary map is then watermarked on the least significant bit of each RGB pixel of the camera sensor's stream, thus obtaining an efficient encoding of the keypoint information and the color data within the same image stream. This encoded RGB frame is then trasmitted along with the depth information through either a MIPI or HDMI interface (to preserve low latency) or to a single USB 3.0 output, obtaining a real time *RGB-KD* (RGB, Keypoints and Depth) streaming device.

From the host side, it is only required to decode the watermark information from the raw camera stream in order to access the list of detected keypoints directly from the *RGB-KD* stream. This solution entirely removes the keypoint detection computational cost from the host, which is function of $w \times h$, allowing to skip directly to the descriptor computation which is instead function of k where k is the number of pre-detected keypoints. A variety of detectors can be instantiated *on-the-fly* within KCNN by uploading to the internal memory the relative weights through the USB interface.

8. Conclusion

A novel, compact convolutional neural network for keypoint detection has been introduced in this paper. The network is used for approximating existing keypoint detectors without the need for changing the underlying implementation, by retraining it according to the specific task. This approach allows different detectors to be easily deployed in a variety of different platforms which span from GPUs to dedicated hardware logic for neural networks.

Together with a novel training set reinforcement approach which exploits the knowledge of the function to be approximated, the network successfully learns the state of the art detector, *i.e.*, KAZE. Results show that the detection performance exceeds the one of SIFT and of the cutting edge machine-learning based TILDE and LIFT. The hardware implementation of the proposed architecture is able to deliver extremely low latency and low power keypoint detection, paving the way to mainstream applications such as AR/VR and autonomous driving as well as embedded systems.

References

[1] H. Aanæs, A. Dahl, and K. Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97:18–35, 2012.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*, abs/1603.04467, 2016.

[3] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. KAZE Features. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV'12, pages 214–227, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] H. Altwaijry, A. Veit, and S. Belongie. Learning to detect and match keypoints with deep architectures. In *British Machine Vision Conference (BMVC)*, York, UK, 2016.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[6] G. Bradski. OpenCV. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] P. Dollar, Z. Tu, and S. Belongie. Supervised Learning of Edges and Object Boundaries. In *Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 1964–1971, Washington, DC, USA, 2006. IEEE Computer Society.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep Learning with Limited Numerical Precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1737–1746. JMLR Workshop and Conference Proceedings, 2015.

[9] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 6 2000.

[10] W. Hartmann, M. Havlena, and K. Schindler. Predicting Matchability. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 9–16, June 2014.

[11] T. Lindeberg. Image Matching Using Generalized Scale-Space Interest Points. *Journal of Mathematical Imaging and Vision*, 52(1):3–36, 2015.

[12] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[13] P. Mainali, G. Lafruit, K. Tack, L. V. Gool, and R. Lauwereins. Derivative-Based Scale Invariant Image Feature Detector With Error Resilience. *IEEE Transactions on Image Processing*, 23(5):2380–2391, May 2014.

[14] P. Mainali, G. Lafruit, Q. Yang, B. Geelen, L. V. Gool, and R. Lauwereins. SIFER: Scale-Invariant Feature Detector with Error Resilience. *Int. J. Comput. Vision*, 104(2):172–197, Sept. 2013.

[15] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. In *Proceedings of the British Machine Vision Conference*, pages 36.1–36.10. BMVA Press, 2002.

[16] K. Mikolajczyk and C. Schmid. An Affine Invariant Interest Point Detector. In *Proceedings of the 7th European Conference on Computer Vision-Part I*, ECCV '02, pages 128–142, London, UK, UK, 2002. Springer-Verlag.

[17] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A Comparison of Affine Region Detectors. *International Journal of Computer Vision*, 65(1):43–72, 2005.

[18] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53, Mar. 2008.

[19] A. Richardson and E. Olson. Learning convolutional filters for interest point detection. In *2013 IEEE International Conference on Robotics and Automation*, pages 631–637, May 2013.

[20] E. Rosten and T. Drummond. Machine Learning for High-speed Corner Detection. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[22] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua. Learning Separable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):94–106, Jan 2015.

[23] J. Šochman and J. Matas. Learning Fast Emulators of Binary Decision Processes. *International Journal of Computer Vision*, 83(2):149–163, 2009.

[24] C. Strecha, A. Lindner, K. Ali, and P. Fua. Training for Task Specific Keypoint Detection. In *Pattern Recognition, 31st DAGM Symposium, Jena, Germany, September 9-11, 2009. Proceedings*, pages 151–160, 2009.

[25] L. Trujillo and G. Olague. Using Evolution to Learn How to Perform Interest Point Detection. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 1, pages 211–214, 2006.

[26] Y. Verdie, K. M. Yi, P. Fua, and V. Lepetit. TILDE: A Temporally Invariant Learned DEtector. In *Proceedings of the Computer Vision and Pattern Recognition*, 2015.

[27] K. Wilson and N. Snavely. Robust Global Translations with 1DSfM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[28] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. In *Proceedings of the European Conference on Computer Vision*, 2016.