Design and Deployment of an Image Polarity Detector with Visual Attention

Edoardo Ragusa¹ · Tommaso Apicella¹ · Christian Gianoglio¹ · Rodolfo Zunino¹ · Paolo Gastaldo¹

Received: 12 June 2020 / Accepted: 12 January 2021 / Published online: 24 January 2021
© The Author(s) 2021

Abstract

Embedding the ability of sentiment analysis in smart devices is especially challenging because sentiment analysis relies on deep neural networks, in particular, convolutional neural networks. The paper presents a novel hardware-friendly detector of image polarity, enhanced with the ability of saliency detection. The approach stems from a hardware-oriented design process, which trades off prediction accuracy and computational resources. The eventual solution combines lightweight deep-learning architectures and post-training quantization. Experimental results on standard benchmarks confirmed that the design strategy can infer automatically the salient parts and the polarity of an image with high accuracy. Saliency-based solutions in the literature prove impractical due to their considerable computational costs; the paper shows that the novel design strategy can deploy and perform successfully on a variety of commercial smartphones, yielding real-time performances.

Keywords Image polarity detection · Convolutional neural network · Embedded systems

Introduction

The availability of ever-increasing computational power and the diffusion of distributed computing allow to apply deep-learning paradigms to complex problems. Deep learning also lies at the core of a variety of applications supported by smart devices, but power consumption and hardware constraints tend to limit the deployment of those learning models. Convolutional neural networks (CNNs), for example, represent a key tool to deal with image/video processing domains, but convey a notable effort in the architecture design and bring about a considerable computational cost.

This makes the real-time implementation of CNNs on embedded systems a very challenging task. In practice, when addressing resource-constrained devices, a trade-off between generalization accuracy and latency is of paramount importance.

Sentiment analysis is a most interesting, yet challenging application relying on deep learning [1, 2], since it aims to extract the emotional information conveyed by media contents. That problem calls for a multidisciplinary approach, involving cognitive models [3], computational resources [4], Natural Language Processing [5, 6], and multimodal analysis [7]. Image polarity detection, in particular, deals with the emotional information conveyed by an image, which even worsens the complexity of sentiment analysis. To tackle the so-called *subjective perception problem* [8], i.e., different users perceive the same image in different ways, designers often envision custom solutions, involving both algorithms and hardware implementations.

Modern polarity detectors rely on CNNs, mostly because that paradigm proved very effective in extracting suitable features from images [9]. Moreover, CNNs can support saliency detection (to identify the parts of an image that draw an observer's attention). This is crucial for polarity detection [10–12], since the salient parts of an image carry relevant information for sentiment analysis, too. In spite of

✉ Edoardo Ragusa
edoardo.ragusa@edu.unige.it

Tommaso Apicella
s4111548@studenti.unige.it

Christian Gianoglio
christian.gianoglio@edu.unige.it

Rodolfo Zunino
rodolfo.zunino@unige.it

Paolo Gastaldo
paolo.gastaldo@unige.it

¹ Department of Electrical, Electronic and Telecommunications Engineering and Naval Architecture, DITEN, University of Genoa, Genoa, Italy

these promising features, CNNs cannot usually fit the constraints imposed by smart devices [13].

To tackle the latter issue one might consider cloud computing, and move the computational load from a user's device to remote data centers. That approach, however, brings about significant side effects [14]: firstly, fast connections and high-power servers should always be available; more importantly, privacy concerns might arise, when considering that a user's emotional perception can relate to religion, sexual, and political orientations. The implementation of the inference, forward phase in a user's device therefore seems a viable solution. Since battery-operated systems cover the majority of smart devices, energy consumption is a crucial design constraint when targeting real-world applications.

This paper extends previous research [15], and presents a design strategy to deploy image polarity detection tools on resource-constrained embedded systems; it introduces implementation details, an extensive analysis about hardware options, and new experiments. Mobile devices are the target platforms, hosting on-line polarity detectors supported by hardware-oriented deep neural networks. Experimental results confirm that commercially available smartphones can support image polarity detection endowed with visual attention capabilities, thus opening new vistas on the development of low-cost applications in the man–machine interaction domain.

Contribution

The continuous introduction of new, more powerful mobile devices emphasizes the need to trade-off generalization performances and hardware consumption. The widespread application of deep-learning solutions has further widened the gap between theoretical algorithms and real-time, hardware-constrained implementations. A most frequent option for deployment is to remove the critical functional blocks of the inference systems altogether. Saliency detection, which is commonly embedded in sentiment classifiers, makes no exception and, to the authors' best knowledge, has never been implemented in resource-constrained devices. The original contribution of the paper can be outlined as follows:

- The definition of a hardware-friendly Saliency Detector, relying on a hardware-friendly deep neural network and suitably designed to fit embedded and constrained systems.
- As a result, any embedded implementation of image sentiment analysis can now benefit from saliency detection. When tested on standard benchmarks, this solution

proved outperformed existing implementation of polarity detectors for embedded systems.

- A complete design strategy aimed to the deployment on commercial smartphones. The process takes into account the definition of the algorithm, the quantization mechanisms with the associate effects, and the required tools for the real-time deployment.

The paper is organized as follows. Section 2 reviews the state-of-the-art in the literature. Section 3 presents the hardware-aware deep-learning models that support the proposed solution, whereas Section 4 introduces and analyzes the novel algorithm. Section 5 describes the design aspects to ensure an effective deployment on mainstream embedded devices. Section 6 presents the experimental results obtained, and Section 7 makes some concluding remarks.

Related Works

Today's state-of-the-art implementations of image polarity detection all rely on deep learning. An exhaustive review of the several proposed approaches can be found in [7–9].

The approaches to polarity detection typically rely on low-level features, which are extracted by a CNN (pre-)trained for object recognition; a fine-tuning process adjusts the network parameters for the specific application task. The various methods mostly differ in: a) the adopted CNN, b) the transfer-learning technique, and c) the training data domain. Campos et al. [16] adopted the *AlexNet* architecture for feature extraction, and analyzed the effects of layer ablation/addition on the eventual generalization performances of the fine-tuned network. Ontology-based representations have been widely used on top of object recognition models [17]; You et al. [18] introduced an architecture in which a pair of convolutional layers and four fully connected layers were stacked.

Recent research aimed to augment the basic, fully convolutional architectures. In some variants for polarity detection, both the image and a set of additional features formed the network inputs. The *Vgg_19* architecture proposed by Fan et al. [10] processed combined image and focal-channel information to model human attention. Likewise, other works studied the role played by art features [19], context, and information [20]. Qian et al. [21] proposed a combination of CNNs and recurrent neural networks to model sequence of different sentiments in one image.

Several papers recently tackled the saliency issue: in [11], a specific detector [22] pinpointed the relevant parts of an image. Likewise, in [12] local information supplied by a Saliency Detector and global features extracted from the

entire image merged in a fusion step. Recently, Wu et al [23] proposed a multi-task learning algorithm that integrated saliency detection and attention mechanisms into a deep recurrent network architecture. Attention mechanisms, as defined in [24], were also explored in [25], where a dedicated convolutional neural network extracted context information from the original input image. Similarly, [26–28] applied attention mechanism, whereas Rao et al. [29] adopted a faster-RCNN to locate the significant parts of an image.

The literature proves that multimodal approaches can enhance the performances of sentiment classifiers [30]. Those models, however, depend critically on the performances of the feature extractors when addressing different information sources, and require an additional computation effort to combine the extracted features.

The deployment of image polarity classifiers on resource-constrained devices seems to have drawn a limited attention. In [13], the authors analyzed solutions including hardware-friendly neural networks and weight truncation, but considered quite expensive hardware accelerators for deep learning. By contrast, this paper targets standard microprocessors and extends that research by including automatic saliency detection, to the purpose of further enhancing visual-polarity assessment.

Hardware-Friendly Deep Networks

Deep networks allow for different architectures, not all of them prove suitable when targeting hardware implementations. In practice, one needs to properly balance accuracy, memory consumption, and computational cost.

The development of an image polarity detector endowed with saliency-detection capability involves two main tasks of computer vision, i.e., object classification and detection. The following sections briefly review the existing solutions to trade-off the generalization performances and the computational costs of the models.

Object Classification

From among the proposed hardware-oriented solutions for object classification, the family of MobileNets represents a most popular approach. The depth-wise separable convolution (DSC) is a key feature of MobilenetV1 [31–33]. In DSC, the standard convolutional operator is replaced by a pair of separate layers, which implement a factorized version of the original operation. The first layer supports a depth-wise convolution and involves one convolutional filter per input channel. The second layer is a 1×1 , point-wise convolution, which extracts a new set of features by

working out linear combinations of the input channels. Such a decomposition remarkably reduces computational costs. For an input set of size $H \times W \times M$, and a convolutional layer characterized by N kernels of size $D_k \times D_k$, the computational cost C_{sc} of the conventional convolution is

$$C_{sc} = H \times W \times M \times N \times D_k \times D_k. \quad (1)$$

When using the factorized version, the associate cost C_{DSC} is

$$C_{DSC} = H \times W \times M \times (D_k^2 + N). \quad (2)$$

which is significantly smaller than (1).

Recently, Ragusa et al. [13] showed that, in the presence of a large dataset for fine-tuning, one can implement polarity detection by means of hardware-friendly CNNs without compromising the eventual generalization performances. Those results were achieved by means of *MobileNetV1*, an architecture specifically designed for embedded systems.

Object Detection

Not all parts of an image are equally important, and a detection process is often useful to identify the relevant sub-sections. Box-based solutions for saliency detection can be divided into region-based and single-shot detectors.

The former approaches rely on a region-cropping network that first detects the informative parts of the image [34, 35], then evaluates the visual contents by treating each sub-region as an independent image. This model attains state-of-the-art performances in terms of accuracy, but typically cannot satisfy hardware constraints. Furthermore, the inference time grows linearly with the number of detected boxes.

Single-shot detectors [36, 37] (SSDs) merge the region cropping and classification steps into one operation. The architecture uses a set of pre-set boxes called anchors. By using anchors, the network need not select the regions of interest a priori, but computes class probabilities and the positions of all anchors in one step. The resulting speed-up in the inference phase might be counterbalanced by a loss in accuracy, especially in the identification of small objects.

Several CNN architectures involving DSC can attain state-of-the-art performances in object detection [31–33, 38, 39]. Most Saliency Detectors derive meta-architectures [22, 23, 40] from object detection. According to [41], the framework combining MobilenetV1 and Single-Shot Detector seems the best option for implementing object detection on resource-constrained devices. An empirical analysis considering latency vs accuracy involved more than 100 different configurations and proved that the combined setup lied on the Pareto-optimal surface [41].

Fig. 1 Complete schema showing the classification flow for the whole system

The Proposed System

The paper adopts a bottom-up approach to the enhancement of image polarity detection, which splits the problem into sub-tasks [12]. Each module of the described architecture is designed by taking into account possibly hard constraints in terms of memory usage, power consumption, and the number of floating-point operations.

Figure 1 outlines the overall processing flow. The system includes four logical blocks and is organized into two branches. The upper branch implements a conventional classification scheme: the entire input image feeds a classifier, that works out the associate polarity (P_{tot}). In the lower branch, a saliency-detector block first processes the input image, and yields a set of sub-regions holding the salient parts in the image itself. The sub-regions feed a Patch Classifier, which assigns a polarity estimate to each of them. A Fusion Block eventually combines the outcomes of both classifiers and works out the overall polarity label associated with the input image.

In the following, it is assumed that the output of each polarity classifier is a scalar quantity $\in [-1, 1]$, where -1 means totally negative, and $+1$ means totally positive.

Image Classifier

In the Image Classifier, one CNN is trained to estimate image polarity; this module processes the input image as a whole without any additional information. A pre-trained CNN architecture provides the starting point, while a fine-tuning procedure adjusts the image polarity detection process.

The design strategy adopts MobileNetV1 as the core CNN. The architecture is characterized by 4.2M parameters, and each inference (forward) phase requires 569M operations. It

is worth noting that the most popular architecture for image polarity, namely VGG_16, involves 138M parameters and requires 15300M operations to carry out an inference step [32]. MobilenetV1 is about 32 times lighter in terms of memory usage, and performs 27 times more efficiently in terms of floating-point operations. Nonetheless, MobileNetV1 attains a comparable accuracy in polarity detection [13].

Saliency Detector

Saliency detection locates the significant parts of an image that are more interesting for a human viewer, and typically relies on object-detection approaches [12] [41]. Deep networks typically split the inference strategy for object detection into two phases: 1) a meta-architecture identifies a set of “boxes” delimiting the tentative sub-regions, and 2) a backbone CNN extracts features from each box. A pre-trained network for object detection again provides the starting point for training, followed by a fine-tuning procedure specifically targeted to classify the saliency of each sub-region.

In the design schema presented here, the Saliency Detector relies on an SSD-MobileNetV1 meta-architecture, which combines both box identification and box cropping and classification, thereby reducing the overall computational cost. The process requires 1.2 billion operations and storage for 6.8 M parameters. The equivalent process using a Fast R-CNN as an alternate meta-architecture and VGG_16 as a feature extractor, would involve 64.3 billion operations and 138.5M parameters.

Patch Classifier

The Patch Classifier assigns a polarity to each sub-region prompted by the Saliency Detector. This module adopts the

same structure of the Image Classifier, but the fine-tuning procedure covers the sub-regions extracted from image polarity datasets. Section 6.2 will provide further details of this procedure.

Fusion Block

The Fusion Block handles the outcomes of the Image Classifier and the Patch Classifier to associate a polarity estimate to the input image. This module implements a hard-coded algorithm and is not subject to training. The rationale underlying the algorithm is that bigger objects in an image often tend to draw the viewer’s attention and therefore prove to be more important. In fact, small objects such as guns, injuries, or particular symbols can sometimes be crucial. The Fusion Block balances these two aspects by clustering the sub-regions into groups based on their size. Then, the overall polarity is assessed. The procedure can be outlined as follows:

- Salient sub-regions are grouped into 5 clusters $C_j, j = 1, \dots, 5$ on the basis of the ratio of the area of the i -th sub-regions A_i to the area of the entire image A_{tot} . The five ranges are: $(0.0, 0.2)$, $[0.2, 0.4)$, $[0.4, 0.6)$, $[0.6, 0.8)$, $[0.8, 1)$. For instance, the set C_1 includes the patches such that $0.0 < A_i/A_{tot} < 0.2$.
- The cumulative polarity of each cluster is worked out by adding the polarities of all cluster elements: $P_{C_i} = \sum_{j \in C_i} P_j$.
- The eventual polarity is computed by picking the element featuring the maximum absolute value in the vector $h = [P_{C_1}, P_{C_2}, P_{C_3}, P_{C_4}, P_{C_5}, P_{tot}]$. Let $\hat{P} = \arg \max_j (|h_j|)$; then, one assigns positive or negative polarity according to the sign of $\hat{P}$. For instance, if an image is characterized by $h = [-0.1, -0.7, -0.3, +0.5, -0.2, +0.3]$, then $\hat{P} = -0.7$ and the eventual image polarity is negative.

Deployment on Embedded Devices

The deployment of any deep-learning algorithm on embedded devices should take into account some constraints. First, the target device should properly balance memory size, power consumption, computational capacity and cost. Secondly, data types (in terms of bit representation) should be carefully selected, as they impact on the trade-off between accuracy and memory usage.

Devices

Conventional deep-learning algorithms require GBs of memory to be deployed. For this reason, the market offers

embedded hardware accelerators with dedicated memories. Intel released the “Movidius” neural-computing stick that includes 8GBs of memory for parameters storage and inference operations. Nvidia offers the “Jetson” family, in which microcontrollers are embedded side by side with a dedicated GPU according to the “Systems-On-Modules” paradigm. Google’s “Coral” line of edge embedded devices relies on Tensor Processing Units (TPUs) for the inference phase. All those solutions leverage dedicated hardware to run deep-learning algorithms for limiting latency. Hardware accelerators also involve advanced software tools, such as sophisticated compression strategies to reduce storage requirements. A typical trick is to apply a fixed-point representation whenever required. Ragusa et al. [13] recently deployed image sentiment classifiers on Movidius and Jetson devices; that paper proved empirically that real-time performances could be obtained without affecting the accuracy of the eventual predictors.

In many cases, however, application-specific constraints force designers to look for different solutions. Most commercial products, including smartphones and smart devices, set tight constraints on power consumption. Although recent high-end smartphones embed Graphical Processing Units (GPUs) and Neural Processing Units (NPU) [42], those devices are expensive and quite demanding in terms of power consumption. Moreover, even in the presence of a memory ranging in GBs, one application can only use a limited portion of it.

Smart devices often rely on microcontrollers; hence, the quantity of available memory is limited to few MBs. For instance, the ARM Cortex M7—largely embedded in IoT applications—just reserves 2 MBs for parameter storage. External RAMs and flash memories can extend memory space, but these add-ons must keep relatively small to limit power consumption and make any design strategy based on embedded accelerators impractical.

Data Representation

Low-level programming languages can improve the efficiency in the implementation of deep neural networks, but at the same time their adoption can increase a product’s Time-To-Market significantly. Thus, designers often rely on high-level programming languages that simplify optimization by means of dedicated libraries and tools, and make large use of high-precision data representations.

Post-training quantization is a common approach to deploy the resulting deep-learning models on resource-constrained devices after training. This solution consists in converting the representation of the network parameters from single-precision floating-point (FP32) to half-precision floating point (FP16), or even 8-bit integer (INT8), coding. FP32 complies with the IEEE floating-point standard and

does not prove critical from an accuracy point of view. Many devices, however, inherently support FP16 in their instruction set. In addition to halving memory occupation, this also yields a considerable speedup in inference operations. For example, high-end microcontrollers typically support the Single Instruction Multiple Data (SIMD) paradigm with the FP16 format.

The INT8 format further reduces memory occupation, and makes fixed-point operations faster, to the detriment of dynamic range and resolution. To minimize the consequent loss in accuracy, a pair of factors modulate integer post-quantization as follows:

$$real = (approx8_{bit} - bias) * scale \quad (3)$$

where *bias* is an integer quantity that sets the center value of represented numbers, and *scale* is a floating-point 32-bit value that re-calibrates the dynamic range.

An INT8 representation calls for some steps for floating-point re-calibration. These steps avoid that rounding issues cause data losses when data propagate through the architecture. Even in the presence of scaling and biasing, 8-bit quantization can affect a predictor accuracy significantly. At the same time, most commercial tools do not provide software libraries supporting optimization for 8/16 bit data; hence, the model parameters are often converted to a floating-point representation. As a result, the adoption of INT8 ultimately does not reduce the run-time memory occupation.

Design Choices

The deployment of the enhanced architecture for image sentiment classification with saliency detection includes a variety of techniques. DSC replaces basic convolutions, and the MobileNetV1 architecture is applied, as a result of the analysis described previously [9]. The detection architectures embed Single-Shot Detectors, with post-training weight quantization. To comply with hardware constraints, the quantization mechanism only affects parameter storage, whereas 32-bit floating-point precision is restored and kept during inference operations.

The set of embedded devices used to test the design criterion included five (Android) smartphones, namely, Honor 10, Huawei P10, Samsung S8, Huawei P8 Lite, and Huawei P20 Lite. This choice was justified by the fact that smartphones are the mainstream from among battery-operated devices. Although most of these devices embed GPUs and NPUs, only the microcontroller components therein were used to work out computations, to ensure fair comparisons. The analysis also excluded multi-threading for the same reason. Measuring performances on a single core emulated a maximally constrained scenario. The experiments were carried out by implementing a custom Android application.

The TFLite tool supported the deployment of the design strategy on the target smartphones; the network weights were only quantized on the host disk. During the inference phases, instead, the weights were cast to FP32 precision after being uploaded into the RAM of each device. Thus, a model quantized in the INT8 format could require 1/4 of disk space as compared with RAM occupation. In the rest of the paper, quantization will only apply to disk-based storage and therefore will not affect the inference phase.

Experimental Results

The experimental session aimed both to evaluate the accuracy of the proposed method and to assess memory usage and inference timings.

The experimental results are organized as follows. Sections 6.1 and 6.2 present the accuracy and memory occupation of the Saliency Detector and the pair {Image Classifier, Patches Classifier}, respectively. Section 6.3 addresses the performances of the whole framework, whereas Section 6.4 illustrates the obtained results in a graphic form. The subsequent experimental campaign analyzed the impact of post-training quantization on the generalization performances. A final experimental session measured the hardware performances of the classifier when deployed on the target smartphones.

Saliency Detector

The training of the Saliency Detector adopted the procedure proposed in [12]. The training involved an SSD_MobileNet architecture trained for object detection by using the COCO dataset [43]¹. The configuration file of the original model, as downloaded from the tensorflow website, sets all the training parameters, with the exception of the number of iterations, which was set to 50,000.

The experiments involved a pair of (publicly available) datasets, holding images with the associate information about salient contents. The starting model was first adjusted by using the ILSVRC-2014 dataset, which contained 127030 images. Then, a fine-tuning process involved the SOS dataset [8], which included 3951 images. The generalization performance was assessed by applying the conventional hold-out procedure: in each experiment, 90% of the available data formed the training set, whereas the remaining 10% of the data provided a test set, which was never used to tune any parameter (during training) or hyper-parameter (during model selection).

¹ https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Table 1 Saliency-detector performances tested on 13982 true boxes

Saliency detector	Predicted	IoU > 75%	Precision
ILSVRC	13118	10207	77.8
SOS	14723	9272	63.0

Table 1 gives the results of the experimental session. The first column lists the different Saliency Detectors that were tested: “ILSVRC” marks the model trained with that dataset, while “SOS” refers to the model first trained with ILSVRC, then refined with SOS. The second column gives the number of boxes prompted by the model. The third column gives the IoU75 mAP indicator [43], used for assessing the object-detection performance. The general Intersection over Union (IoU) measure takes into account the set A of pixels associated with a box proposed by the detector and the ground-truth set of pixels B . It can be formalized as:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}. \quad (4)$$

Thus, IoU75 means that a prediction is classified as correct if $IoU(A, B) > 0.75$. Informally speaking, this quantity only counts as correct predictions those outcomes that are sufficiently overlapped with respect to the real position of the salient region.

The rightmost column reports on the precision, i.e., the ratio of the number of correct predictions to the total number of predictions.

The outcomes of this experimental session confirmed that the number of predicted boxes for the test set was largely consistent with the ground-truth (13982). Moreover, the IoU measurements proved that the predicted boxes overlapped the actual positions satisfactorily. IoU exceeded the 75% threshold in the 77.8% of the cases for SSD_MobileNet_ILSVRC, and in the 63.0% of cases for SSD_MobileNet_SOS. These results seem quite interesting when considering the severe hardware constraints on the architecture.

Image Classifier and Patch Classifier

The subsequent experimental session measured the performances of the pair of classification modules, both relying on the MobileNetV1 architecture.

Training the CNN for image polarity detection required a large dataset. Toward that end, the training sessions used a version of the adjective–noun pairs (ANP) dataset [44] (images crawled from Flickr). The experiments involved the ANP40 dataset [9], which covered the 20 most positive and the 20 most negative adjective–noun pairs. The associate set of images was split into the corresponding subsets (positive and negative samples). The eventual ANP40 set held 10,516 image samples.

The Image Classifier and the Patch Classifier shared the same architecture, i.e., the MobileNetV1 model trained on the ILSVRC-2012-CLS dataset², but were fine-tuned separately. The Image Classifier was adjusted on the whole ANP40 dataset; instead, the Patch Classifier was only fine-tuned on the sub-regions extracted from the ANP40 dataset. In the following,

- PC_ILSVRC will refer to the Patch Classifier fine-tuned on the sub-regions extracted by SSD_MobileNet_ILSVRC
- PC_SOS will refer to the Patches Classifier fine-tuned on the patches extracted via SSD_MobileNet_SOS (as per Sec. 6.1).

The well-known datasets Twitter (Tw) [18] and Multi-view (MVSA_eq and MVSA_maj) [45] formed the test sets to assess the classifiers performances. In both cases, a pool of human users had manually annotated the two datasets. Neither parameter tuning nor model selection ever involved these data.

In the case of Tw, the “all-agreed” version was considered. The experimental sessions involved two versions of MVSA: MVSA_eq (i.e., the all-agreed version of MVSA), and MVSA_maj, in which human annotators assigned the eventual label on a majority basis.

To assess the classifiers performances, the procedure first extracted the sub-regions from each test image. Then, the sub-regions were grouped into five clusters according to their relative size (as per Sec. 4.4). Finally, each cluster included both the sub-regions and the entire, original images.

The obtained results are reported in the form of tables. Each table gives the outcomes of the tests on a specific dataset and a configuration of the Patch Classifier. Each table compares the Image Classifier and the Patch Classifier (first column). The second column indicates the possible classifier inputs, i.e., either the set of sub-regions or the set of images. Columns [3–7] refer to the five clusters, grouped according to the coverage spanned by the sub-regions. For a specific classifier and a specific input (i.e., for a specific row of the table), the Column associated with a Cluster gives the average classification accuracy—in the range [0, 1]—for the patches/images belonging to that Cluster.

For instance, in Table 2, C_1 takes into account the sub-regions covering a percentage of the whole image in the range {0%, 20%}. The cell value 0.628 shows that a percentage of 62.8% of the covered sub-regions was correctly classified.

² <https://github.com/tensorflow/models/tree/master/research/slim>

Table 2 Comparison of classification accuracy of Image Classifier and Patches Classifier for dataset Twitter with patch extractor trained on ILSVRC

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image Classif.	Patches	0.628	0.718	0.516	0.768	0.740
	Images	0.718	0.732	0.645	0.750	0.719
PC_ILSVRC	Patches	0.679	0.761	0.645	0.696	0.719
	Images	0.756	0.723	0.726	0.732	0.688

Table 2 gives the outcomes of the tests involving the Tw dataset; in those experiments, the Patch Classifier was supported by PC_ILSVRC.

PC_ILSVRC always outperformed the Image Classifier when small- and medium-sized patches were classified (Clusters C_1 , C_2 and C_3), independently of the classifiers' inputs (either the sub-regions or the whole image). Conversely, the Image Classifier performed better than PC_ILSVRC, for both sub-regions and full images inputs, for C_4 and C_5 . That behavior was expected because larger sub-regions better approximated full images in size and contents.

Table 3 gives the results for the Tw dataset when the sub-regions were extracted from Saliency Detector trained on SOS.

Table 3 Comparison of classification accuracy of Image Classifier and Patches Classifier for dataset Twitter with patch extractor trained on SOS

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image Classifier	Patches	0.584	0.748	0.748	0.685	0.761
	Image	0.734	0.670	0.696	0.710	0.746
PC_SOS	Patches	0.659	0.709	0.738	0.710	0.720
	Images	0.676	0.688	0.734	0.710	0.756

When image sub-regions were the network inputs, PC_SOS scored better accuracy in C_1 and C_4 . In agreement with the results shown in Table 2, the Patch Classifier was the best option in the case of the smallest sub-regions (C_1). In almost other cases, the basic Image Classifier seemed to perform better.

Table 4 reports on the result for the MVSA_eq dataset when adopting PC_ILSVRC, which differed significantly from those obtained on the Twitter dataset. The image-based classifiers always performed better, independently of any inputs provided. Such a behavior was probably due to the fact that the dataset included images with heterogeneous

contents, including text, schemes, draws. As a consequence, a Patch Classifier trained on saliency regions would prove suitable to handle that kind of data.

Table 5 relates to the experiments on MVSA_eq using PC_SOS. The experimental outcomes confirmed the trend shown with PC_ILSVRC. Tables 6 and 7 complete the experimental analysis.

The experimental verification pointed out that the Patch Classifier and the Image Classifier scored comparable accuracy values; differences depended on the processed dataset; the combination of these two algorithms could enhance performances in most cases, and the results from this approach will be reported in the next section.

The Overall Framework

The third experimental session evaluated the performances of the combined framework (Fig. 1). The Saliency Detector, the Image Classifier, and the Patch Classifier were trained as described above. The Tw, MVSA_eq, and MVSA_maj datasets formed the test sets, and they were never used for either model selection or parameters tuning. A pair of alternative configurations provided the comparisons for the experimental analysis.

MobileNetV1 was the first comparison, i.e., an image polarity detector that did not include any saliency-based technique. This model was selected because 1) it was

Table 4 Comparison of classification accuracy of Image Classifier and Patch Classifier for dataset MVSA_eq with patch detector trained on ILSVRC

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image Classifier	Patches	0.519	0.578	0.665	0.649	0.631
	Image	0.722	0.689	0.636	0.649	0.614
PC_ILSVRC	Patches	0.839	0.852	0.799	0.814	0.817
	Images	0.829	0.852	0.804	0.818	0.805

Table 5 Comparison of classification accuracy of Image Classifier and Patch Classifier for dataset MVSA_eq with patch extractor trained on SOS

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image	Patches	0.624	0.635	0.669	0.656	0.605
Classifier	Image	0.701	0.710	0.689	0.625	0.611
PC_SOS	Patches	0.903	0.863	0.834	0.831	0.744
	Images	0.891	0.859	0.836	0.820	0.748

most efficient in terms of computational cost, and 2) it achieved the best classification accuracy from among a set of candidate architectures on a very similar experimental setup [13].

The second comparison will be denoted in the following as *FullyDD*, and relied on a fully data-driven strategy based on the proposed framework (Fig. 1); a linear regression module supported the Fusion Block. The value of the regularization parameter was set by using a fivefold cross-validation procedure.

Table 8 refers to the Twitter dataset. The first column lists the various predictors, namely, MobileNetV1, FullyDD including the Patch Classifier trained on SOS, FullyDD with the Patch Classifier trained on ILSVRC, the proposed framework with the Patch Classifier trained on SOS, and

Visual Examples

This section gives some visual examples that illustrate the behavior of the proposed framework. Figure 2 includes nine test images. Each sample is outlined in red/green when it was associated with a negative/positive polarity. In each image, a box highlights the salient part as pinpointed by the Saliency Detector, with the same color scheme.

The tests involved different sizes of the patches. In all cases, the overall polarity of the image matched the polarity assigned to the salient sub-region, with the only exception of the leftmost example in the central row. The latter example clarifies that the eventual polarity assigned to an image depended on both the relative size of the sub-region and the strength of the polarity value associated

Table 6 Comparison of classification accuracy of Image Classifier and Patch Classifier for dataset MVSA_maj with patch detector trained on ILSVRC

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image	Patches	0.524	0.542	0.627	0.620	0.621
Classifier	Image	0.691	0.627	0.614	0.628	0.599
PC_ILSVRC	Patches	0.811	0.791	0.785	0.774	0.778
	Images	0.812	0.803	0.781	0.785	0.786

the proposed framework with the Patch Classifier trained on ILSVRC. Columns [2–5] give the performances on the test set measured in terms of accuracy, precision, recall, and F1. The experimental results confirmed that both implementations of the proposed framework outperformed the baseline solutions in terms of accuracy, recall, and F1, with the exception of the *FullyData_SOS* implementation.

Tables 9 and 10 show the results for the MVSA_maj and MVSA_eq datasets, respectively, following the same format adopted in Table 8. In both experiments, the proposed implementation always outperformed the baseline solutions. *FullyDD_SOS* only scored a better precision on the MVSA_maj dataset.

with that patch. In that specific case, the absolute value of polarity associated with the whole image was larger than the absolute value of the polarity associated with the sub-region.

Robustness to Compression and Quantization

The experiments reported in this section analyzed the impact of quantization. In addition to the baseline floating-point representation (i.e., FP32), the analysis took into account the quantization levels FP16 and INT8. The setup followed the procedure as per Section 5.2: the models were at first trained by using the FP32 representation, then all

Table 7 Comparison of classification accuracy of Image Classifier and Patch Classifier for dataset MVSA_maj with patch detector trained on SOS

Classifier	Input	C_1	C_2	C_3	C_4	C_5
Image	Patches	0.605	0.631	0.617	0.633	0.562
Classifier	Image	0.680	0.674	0.633	0.612	0.564
PC_SOS	Patches	0.871	0.828	0.784	0.776	0.690
	Images	0.860	0.811	0.782	0.779	0.690

Table 8 Performances of the proposed model for dataset Twitter

Network	Accuracy	Precision	Recall	F1
MobileNetV1	70.86	80.00	74.35	77.07
FullyDD _{SOS}	70.63	80.15	73.67	76.77
FullyDD _{ILSVRC}	70.52	79.56	74.35	76.87
Proposal _{SOS}	71.77	76.35	82.79	79.44
Proposal _{ILSVRC}	71.99	78.99	78.31	78.65

weights were truncated to the target integer format. In the inference phase, the truncated weights were back converted to the FP32 representation. TFLite supported all the inference models.

Figure 3 reports on the results for the Proposal_{SOS} model on the Twitter dataset, following the training/test set strategy already described. The bar graph gives 4 measurements, i.e., accuracy, precision, recall, and F1. The blue, red, and green bars refer to the FP32, FP16, and INT8 representations, respectively. The results confirmed that quantization did not affect the generalization performances significantly. In fact, when considering recall and F1 descriptors, the FP16 representation even outperformed the baseline model involving FP32.

Figures 4 and 5 give the results for the MVSA_maj and MVSA_eq datasets. When dealing with the latter datasets, too, quantization proved beneficial: FP16 quantization improved

Table 9 Performances of the proposed model for dataset MVSA_maj

Network	Accuracy	Precision	Recall	F1
MobileNetV1	0.60	0.91	0.61	0.73
FullyDD _{SOS}	0.59	0.91	0.59	0.71
FullyDD _{ILSVRC}	0.61	0.90	0.62	0.73
Proposal _{SOS}	0.69	0.90	0.72	0.80
Proposal _{ILSVRC}	0.63	0.90	0.64	0.75

accuracy, recall, and F1. Notably, also INT8 representation turned out to be effective. These improvements were possibly due to the fact that truncation acted as a low pass filter on images; hence, the eventual prediction was less influenced by details and more influenced by global features.

Table 11 reports on a worst/best case analysis based on the experiments shown in Figs. 3, 4, and 5. The analysis covered the MobileNetV1 and Proposal_{ILSVRC} models, and included all the combinations of 3 deep-learning architectures, 4 measures, and 3 test sets. This led to a total of 36 configurations for each model. In the table, the row named “best” counts how many times the associate quantization level attained the best score. For example, models quantized with FP16 scored the best result in 24 out of 36 configurations. The row marked as “worst” counts the configurations scoring the worst results.

The major outcome of that analysis was that the FP16 quantization quite often proved more robust than FP32; this

Fig. 2 Examples of images labeled using the proposed algorithm with details about the salient parts

Fig. 3 Experimental results using Proposal_{SOS} model for inference on Twitter dataset

allowed to halve the memory to store the model parameters. These results did not contradict those reported in [13]: in that research, the entire inference process adopted a truncated algebra, bringing about severe propagation issues.

Deployment on Smartphones

This section presents the results of the deployment of the proposed design strategy on 5 smartphones. To ensure fair comparisons, the microcontroller module within each smartphone only carried out the computations, and any dedicated HW (such as GPUs and NPUs) as well as multi-threading was excluded. Measuring all performances on a single core allowed to emulate a maximally constrained scenario.

A custom Android application supported all experiments, using the TFLite tool for deployment, and the tested quantization levels were all considered, i.e., INT8, FP16, and FP32. In Table 12, each row marks a target device, whereas the columns relate to the quantization levels. The table gives the average inference timings (in milliseconds) over 30 images, with the corresponding standard deviation in brackets.

The considerably high inference timings were a consequence of using one core in the computations. Secondly, the smartphones behaved differently when subject to quantization

Table 10 Performances of the proposed model for dataset MVSA_eq

Network	Accuracy	Precision	Recall	F1
MobileNetV1	0.64	0.95	0.64	0.76
FullyDD _{SOS}	0.65	0.95	0.65	0.77
FullyDD _{ILSVRC}	0.64	0.94	0.65	0.77
Proposal _{SOS}	0.74	0.95	0.76	0.84
Proposal _{ILSVRC}	0.67	0.95	0.67	0.79

Fig. 4 Experimental results using Proposal_{SOS} model for inference on MVSA_maj dataset

effects; with the noted exception of Samsung S8, all devices proved faster when using the INT8 representation. That discrepancy should be ascribed possibly to the different chipsets and the memory management by the Android operating systems. The results anyway confirmed that the design strategy for the inference phase could deploy successfully on commercial smartphones even when using a single core, and always scored smaller latency values than 1 second. This allowed the model to run in parallel with other applications in real-life use cases. As compared with these achievements, existing solutions featuring saliency detection for image sentiment analysis all relied on VGG and RESNET backbones, but the memory occupation required by those approaches could not satisfy the hardware constraints of smartphones[13].

Fig. 5 Experimental results using Proposal_{SOS} model for inference on mvsa_eq dataset

Table 11 Worst and best cases analysis

	FP32	FP16	INT8
Best	11	24	1
Worst	7	4	25

Table 12 Inference time measures for 30 images

	INT8	FP16	FP32
Honor 10	349 (58)	464 (94)	527 (104)
Huawei P10	523 (88)	672 (135)	666 (135)
Samsung S8	568 (95)	497 (98)	496 (97)
Huawei P8 Lite	734 (124)	867 (176)	860 (173)
Huawei P20 Lite	673 (158)	820 (176)	817 (191)

Conclusions

The paper explored the use of deep neural networks, based on depth-wise separable convolutions, as the key elements for image polarity detection endowed with saliency information. The overall problem was addressed using a bottom-up approach, and the eventual system included three deep-learning stages based on MobileNetV1. The overall setup proved computationally lighter than the conventional solutions presented in the literature. Experimental results confirmed that the proposed method managed to balance generalization performances and computational costs effectively.

Funding Open Access funding provided by Università degli Studi di Genova

Compliance with Ethical Standards

Conflicts of Interest The authors declare that they have no conflict of interest.

Ethical Standard Informed consent was not required as no human or animal subjects were involved.

Human and Animal Rights This article does not contain any studies with human or animal subjects performed by any of the authors.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Cambria E, Poria S, Gelbukh A, Thelwall M. Sentiment analysis is a big suitcase. *IEEE Intell Syst.* 2017;32(6):74–80.
2. Dashtipour K, Poria S, Hussain A, Cambria E, Hawalah AY, Gelbukh A, Zhou Q. Multilingual sentiment analysis: state of the art and independent comparison of techniques. *Cogn Comput.* 2016;8(4):757–71.
3. Susanto Y, Livingstone AG, Ng BC, Cambria E. The hourglass model revisited. *IEEE Intell Syst.* 2020;35(5):96–102.
4. Cambria E, Li Y, Xing FZ, Poria S, Kwok K. Senticnet 6: Ensemble application of symbolic and subsymbolic AI for sentiment analysis. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*; 2020. p. 105–114.
5. Xia Y, Cambria E, Hussain A, Zhao H. Word polarity disambiguation using bayesian model and opinion-level features. *Cogn Comput.* 2015;7(3):369–80.
6. Akhtar MS, Ekbal A, Cambria E. How intense are you? predicting intensities of emotions and sentiments using stacked ensemble. *IEEE Comput Intell Mag.* 2020;15(1):64–75.
7. Poria S, Cambria E, Bajpai R, Hussain A. A review of affective computing: From unimodal analysis to multimodal fusion. *Information Fusion.* 2017;37:98–125.
8. Zhao S, Ding G, Huang Q, Chua TS, Schuller BW, Keutzer K. Affective image content analysis: A comprehensive survey. In *IJCAI*; 2018. p. 5534–5541.
9. Ragusa E, Cambria E, Zunino R, Gastaldo P. A survey on deep learning in image polarity detection: Balancing generalization performances and computational costs. *Electronics.* 2019;8(7):783.
10. Fan S, Jiang M, Shen Z, Koenig BL, Kankanhalli MS, Zhao Q. The role of visual attention in sentiment prediction. In *Proceedings of the 25th ACM international conference on Multimedia*; 2017. p. 217–225.
11. Zheng H, Chen T, You Q, Luo J. When saliency meets sentiment: Understanding how image content invokes emotion and sentiment. In *2017 IEEE International Conference on Image Processing (ICIP)*; 2017. p. 630–634. IEEE.
12. Wu L, Qi M, Jian M, Zhang H. Visual sentiment analysis by combining global and local information. *Neural Processing Letters.* 2020;51:2063–75.
13. Ragusa E, Gianoglio C, Zunino R, Gastaldo P. Image polarity detection on resource-constrained devices. *IEEE Intell Syst*, pages available online. 2020. <https://doi.org/10.1109/MIS.2020.3011586>.
14. Wang X, Han Y, Leung VC, Niyato D, Yan X, Chen X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun Surv Tutor.* 2020;22(2):869–904.
15. Ragusa E, Apicella T, Gianoglio C, Zunino R, Gastaldo P. An hardware-aware image polarity detector enhanced with visual attention. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020.
16. Campos V, Salvador A, Giro-i Nieto X, Jou B. Diving deep into sentiment: Understanding fine-tuned CNNs for visual sentiment prediction. In *Proceedings of the 1st International Workshop on Affect & Sentiment in Multimedia*; 2015. p. 57–62. ACM.
17. Chen T, Borth D, Darrell T, Chang SF. DeepSentibank: Visual sentiment concept classification with deep convolutional neural networks. *arXiv:1410.8586 [Preprint]*. 2014. Available from: <https://arxiv.org/abs/1410.8586>
18. You Q, Luo J, Jin H, Yang J. Robust image sentiment analysis using progressively trained and domain transferred deep networks. *arXiv:1509.06041 [Preprint]*. 2015. Available from: <https://arxiv.org/abs/1509.06041>

19. Liu X, Li N, Xia Y. Affective image classification by jointly using interpretable art features and semantic annotations. *J Visual Commun Image Represent*. 2019;58:576–88.
20. Balouchian P, Foroosh H. Context-sensitive single-modality image emotion analysis: A unified architecture from dataset construction to CNN classification. In *2018 25th IEEE International Conference on Image Processing (ICIP)*; 2018. p. 1932–1936. IEEE.
21. Qian C, Chaturvedi I, Poria S, Cambria E, Malandri L. Learning visual concepts in images using temporal convolutional networks. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*; 2018. p. 1280–1284. IEEE.
22. Zhang J, Sclaroff S, Lin Z, Shen X, Price B, Mech R. Unconstrained salient object detection via proposal subset optimization. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2016. p. 5733–5742.
23. Wu Z, Meng M, Wu J. Visual sentiment prediction with attribute augmentation and multi-attention mechanism. *Neural Process Lett*. 2020;22:2403–16.
24. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. 2014.
25. You Q, Jin H, Luo J. Visual sentiment analysis by attending on local image regions. In *Proceedings of the thirty-first AAAI conference on artificial intelligence*, 2017. p. 231–237.
26. Yang J, She D, Sun M, Cheng M-M, Rosin PL, Wang L. Visual sentiment prediction based on automatic discovery of affective regions. *IEEE Trans Multimedia*. 2018;20(9):2513–25.
27. Yang J, She D, Lai YK, Rosin PL, Yang MH. Weakly supervised coupled networks for visual sentiment analysis. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2018. p. 7584–7592.
28. Song K, Yao T, Ling Q, Mei T. Boosting image sentiment analysis with visual attention. *Neurocomputing*. 2018;312:218–28.
29. Rao T, Li X, Zhang H, Xu M. Multi-level region-based convolutional neural network for image emotion classification. *Neurocomputing*. 2019;333:429–39.
30. Poria S, Chaturvedi I, Cambria E, Hussain A. Convolutional MKL based multimodal emotion recognition and sentiment analysis. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 439–448. IEEE, 2016.
31. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conf Comput Vis Recognit*, pages 4510–4520, 2018.
32. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861 [Preprint]*. 2017. Available from: <https://arxiv.org/abs/1704.04861>
33. Chollet F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2017. p. 1251–1258.
34. Girshick R. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*; 2015. p. 1440–1448.
35. Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*; 2015. p. 91–99.
36. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2016. p. 779–788.
37. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC. Ssd: Single shot multibox detector. In *European conference on computer vision*, Springer; 2016. p. 21–37.
38. Howard A, Sandler M, Chu G, Chen LC, Chen B, Tan M, Wang W, Zhu Y, Pang R, Vasudevan V, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*; 2019. p. 1314–1324.
39. Zhang X, Zhou X, Lin M, Sun J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2018. p. 6848–6856.
40. Zhang J, Ma S, Sameki M, Sclaroff S, Betke M, Lin Z, Shen X, Price B, Mech R. Salient object subitizing. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2015. p. 4045–4054.
41. Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE Conf Comput Vis Recognit*; 2017. p. 7310–7311.
42. Ignatov A, Timofte R, Chou W, Wang K, Wu M, Hartley T, Van Gool L. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.
43. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL. Microsoft coco: Common objects in context. In *European conference on computer vision*, Springer; 2014. p. 740–755.
44. Borth D, Ji R, Chen T, Breuel T, Chang SF. Large-scale visual sentiment ontology and detectors using adjective noun pairs. In *Proceedings of the 21st ACM international conference on Multimedia*; 2013. p. 223–232.
45. Niu T, Zhu S, Pang L, El Saddik A. Sentiment analysis on multi-view social data. In *International Conference on Multimedia Modeling*, Springer; 2016. p. 15–27.