Compression of convolutional neural networks for high performance image matching tasks on mobile devices

Roy Miles
Imperial College London
r.miles18@imperial.ac.uk

Krystian Mikolajczyk
Imperial College London
k.mikolajczyk@imperial.ac.uk

Abstract

Deep neural networks have demonstrated state-of-the-art performance for feature based image matching through the advent of new large and diverse datasets. However, there has been little work on evaluating the computational cost, model size, and matching accuracy tradeoffs for these models. This paper explicitly addresses these practical constraints by considering the state-of-the-art L2Net architecture. We observe a significant redundancy in the L2Net architecture, which we exploit through the use of depthwise separable layers and an efficient Tucker decomposition. We demonstrate that a combination of these methods is more effective, but still sacrifices the top-end accuracy. We therefore propose the Convolution-Depthwise-Pointwise (CDP) layer, which provides a means of interpolating between the standard and depthwise separable convolutions. With this proposed layer, we are able achieve up to $8\times$ reduction in the number of parameters on the L2Net architecture, $13\times$ reduction in the computational complexity, while sacrificing less than 1% on the overall accuracy across the *HPatches* benchmarks. To further demonstrate the generalisation of this approach, we apply it to the SuperPoint model. We show that CDP layers improves upon the accuracy, while using significantly less parameters and floating point operations for inference.

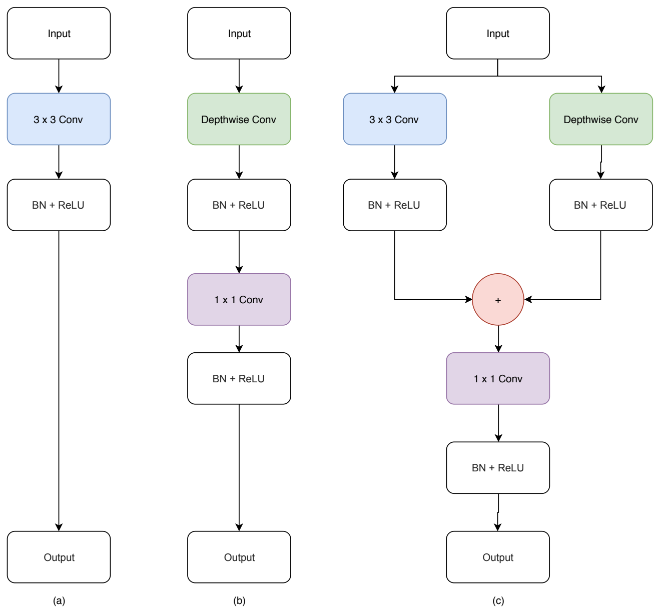

Figure 1: (a) Standard convolutional block. (b) Depthwise seperable layer. (c) Our proposed Convolutional-Depthwise-Pointwise (CDP) layer. The CDP layer aggregates features extracted through a standard and depthwise convolution. Batch normalisation (BN) and ReLU non-linearities are also placed independently after the standard convolution and depthwise convolution stages. The + operator indicates concatenation of tensors across the depth.

1 Introduction

Local features have a wide range of applications in robotics, tracking, or 3D reconstruction. In many such applications, the algorithms are expected to operate in real time on resource constrained devices. However, this is generally not possible for most CNN based models due to the memory and computational cost for inference far exceeding the constraints of the device.

Although the computational cost of handcrafted descriptors has been extensively researched [1; 6; 29; 8], so far there has been no methods improving efficiency of deep-learning based descriptors. In contrast, CNN models for image classification or object detection have been successfully compressed and deployed on mobile platforms through MobileNet [16] or ShuffleNet [40], which has not been possible with large models such as ResNet[14], VGG [32] or GoogLeNet [33].

The model size and computational cost can act as indirect metrics for evaluating the potential deployment on these devices. For example, the inference latency is often limited by the memory and processing speed of the device, which are directly correlated to the model size and theoretical computational cost.

This paper presents the following contributions:

1. We investigate and demonstrate the redundancy of the popular local feature architectures such as L2Net [34] and SuperPoint [10] that can be exploited to meet the practical constraints.
2. We show how depthwise separable layers and tensor decomposition can be used to significantly improve the model performance and how a pointwise linear bottleneck provides a far better accuracy/performance trade-off than applying these methods separately.
3. We propose a novel Convolutional-Depthwise-Pointwise (CDP) layer that allows to linearly interpolate between a standard convolution and an efficient depthwise separable layer.
4. We demonstrate that our solutions for CNN based feature methods can significantly increase the memory and computational efficiency while maintaining the state-of-the-art accuracy on standard benchmarks.

The rest of this paper is organized as follows. In section 2 the related work is revisited, with more details on the methods that our approach is based on. Section 3 presents our proposed approach to compress the L2-Net and SuperPoint models. Section 4 evaluates our approach and discusses the experimental results on a standard benchmark.

2 Related Work

There has been a lot of research on developing handcrafted descriptors that trade-off robustness for computational efficiency. SIFT [22] exploits difference of Gaussian to efficiently extract keypoints. FAST [29] is a simple and effective corner detection approach that compares neighbouring and contrasting pixels in an image. SURF [6] uses a precomputed integral image to accelerate computation of image derivatives at different scales for feature extraction and description. Binary descriptors [8][20] offer a good trade-off between the speed and matching accuracy. However, machine learning approaches have been able to achieve significant accuracy improvements, in particular CNN based methods as recent evaluations [3] show. A number of recently proposed top performing descriptors have used the L2Net [34] architecture with different training methodologies such as HardNet [24], GeoDesc [23], DOAP [13], SOSNet [35], LF-Net[27] etc. Similarly, SuperPoint approach [10] that combines a keypoint detector and descriptor has also been found to perform well in recent evaluations [5]. We therefore focus on L2Net [34] and SuperPoint [10] and demonstrate the improvements by our approach applied on these two architectures.

2.1 L2Net

The L2Net architecture accepts a single grayscale patch as the input, which is subsequently compressed spatially in layers 3, 5, and 7 through the use of strided convolutions. The network uses no bias terms in the convolutional layers or any learnable affine parameters in the batch normalisation layers. The final output is a descriptor of dimensions 1×128, which was chosen to match SIFT and to allow for a direct comparison.

Most of the network parameters are in the last layer due to the relatively large 8×8 receptive field of the kernels and the channel depth. However, as the spatial dimensions of the input feature maps are progressively downsampled through the network, the majority of the computation is in the intermediate layers, specifically layers 2, 4, and 6, which is demonstrated in figure 2.

2.2 SuperPoint

We also analyse the fully-convolutional architecture proposed in SuperPoint [10] for jointly computing keypoints and descriptors.

The SuperPoint model uses a shared VGG backbone that encodes a spatially reduced representation of the input image. This is then fed into two separate decoder networks for computing the interest points and descriptors, respectively. The practical benefit of the SuperPoint pipeline is that it can achieve real-time performance for the entire image matching pipeline. However, its performance was evaluated on a Titan X GPU, which is far from consumer grade hardware or an embedded device. In fact the theoretical performance of these large GPUs is in the order of T(era)FLOPs, whereas for mobile devices this typically is in the range of tens of G(iga)FLOPs - where FLOPs here is described as the number of floating point operations per second. This motivates the exploration into compressing its CNN model, while also demonstrating that our proposed approach can generalise to other models.

2.3 CNN efficiency improvements

This section presents several methods to improve the efficiency of convolutional operations with large tensors that have been successfully exploited in the context of object recognition but not applied to local descriptors yet.

Depthwise separable convolutions. The 2D convolution operation maps an input tensor $\mathcal{X} \in \mathbb{R}^{W \times H \times C}$ to an output tensor $\mathcal{Y} \in \mathbb{R}^{W \times H \times N}$ through the spatial convolution of a 4-way tensor $\mathcal{K} \in \mathbb{R}^{K \times K \times C \times N}$.

$$\mathcal{Y}_{h,w,n} = \sum_{k_1,k_2=1}^{K} \sum_{i=1}^{C} \mathcal{X}_{h',w',i} \cdot \mathcal{K}_{k_1,k_2,i,n} \tag{1}$$

$$h' = (h-1)s + k_1 - p \tag{2}$$

$$w' = (w-1)s + k_2 - p \tag{3}$$

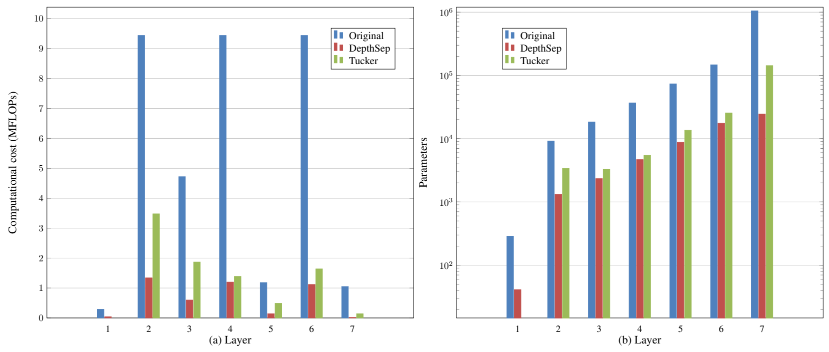

Figure 2: Complexity of the L2Net architecture. Left (a) shows the number of floating point operations in each layer for standard convolutions, depthwise separable layers, and the Tucker decomposition. Right (b) presents the number of parameters on a logarithmic axis.

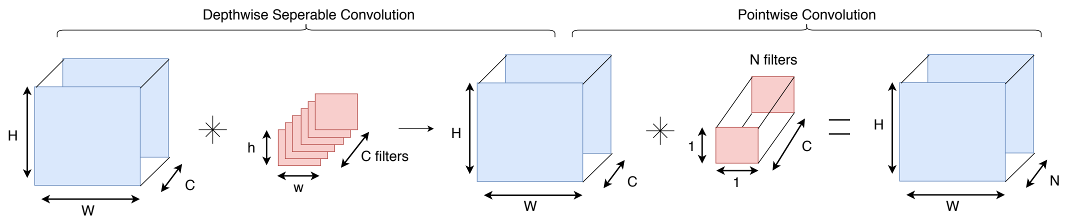

Figure 3: Depthwise separable convolution. Note that each depthwise filter is applied to a single input channel and the results are concatenated to form the output feature map. A pointwise convolution is then used to aggregate these maps and provide dense depthwise connectivity.

where s is the stride and p is the zero-padding size. The standard convolution will have the computational cost of:

$$W \times H \times K^2 \times C \times N \tag{4}$$

In contrast, the depthwise separable layer breaks this operation into two subsequent operations, namely the depthwise convolution, for extracting local spatial features, and the pointwise convolution, for aggregating features across the depth. This can be described through the following formulation:

$$\mathcal{Y}_{h,w,n} = \sum_{i=1}^{C} \sigma(\sum_{k_1,k_2=1}^{K} \mathcal{X}_{h',w',i} \cdot \mathcal{D}_{k_1,k_2,i}) \cdot \mathcal{K}_{i,n} \tag{5}$$

Where the depthwise and pointwise kernels are given by $\mathcal{D} \in \mathbb{R}^{K \times K \times C}$ and $\mathcal{K} \in \mathbb{R}^{C \times N}$ respectively. The intermediate element-wise non-linearity operator is given by σ. Overall, the total computational cost of the depthwise separable layer (omitting the non-linearity) is thus given by:

$$(W \times H \times K^2 \times C) + (W \times H \times C \times N) \tag{6}$$

Depthwise-separable layer was originally proposed in [31] and have since been used in the Inception models [33] and all the MobileNet variants [16] [11]. The original MobileNet [16] also proposed a width multiplier, which can uniformly thin or expand the channels from the depthwise convolution. These layers have been integrated into all the commonly used deep learning frameworks to promote its adoption. Although there will be some inherent accuracy drop by diminishing the dense feature connectivity across the spatial and channel dimensions, this is often not significant in most applications. Figure 3 graphically demonstrates the operation of the depthwise separable layer.

Pruning. Pruning is an active removal of individual weights, kernels, or even entire layers from a network based on a saliency measure or a regularization term. Optimal Brain Damage [39] originally proposed to evaluate the saliency of individual weight entries using an approximation of the Hessian. This idea was further developed in Optimal Brain Surgery [12] through iteratively computing the Hessian to obtain a more exact approximation. [38][21] considers the pruning of weights in a group-wise fashion, which can lead to practical performance improvement without the need for dedicated sparse matrix hardware/software libraries. Pruning based methods for compression are orthogonal to low-rank decomposition, which is explored throughout this paper. In fact, we expect very significant performance improvements can be achieved by further pruning and fine-tuning the models.

Low-rank tensor decomposition. Tensor decomposition is any method for approximating a higher order tensor using simple components, which are then combined using elementary operations. This can lead to a significant reduction in the number of parameters used to represent the original ten-

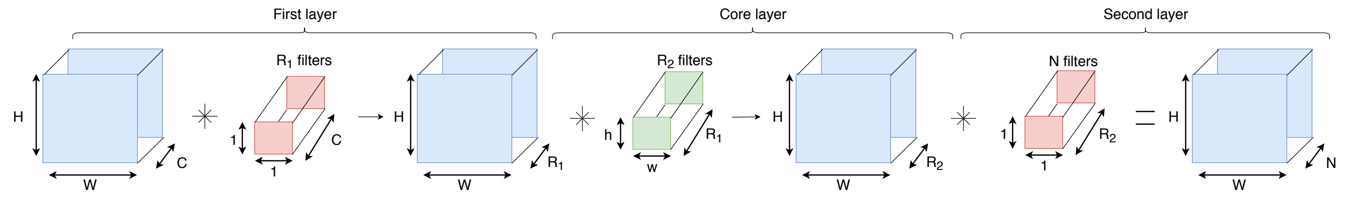

Figure 4: Implementation of the Tucker decomposition [18] applied across the input and output channel modes. The first and last layers can be seen as pointwise projections, while the core tensor provides the spatial aggregation in a lower dimensional subspace. Ranks R_1 and R_2 are determined through VBMF of the original pre-trained weights.

sor, improve computational efficiency, and result in a more compact and interpretable model.

There has been a lot of existing work in factorising existing layers to obtain computational speedup, or to reduce the number of parameters. Tucker [36] and its special case Canonical Polyadic (CP) [15] decompositions are examples that factor a N-dimensional tensor into lower dimensional factors. By using these decompositions, the original convolution operation can then be replaced by a series of convolutions with smaller tensors, which leads to large computational speed-ups [17][18]. Tensor networks further provide a framework for these decomposition methods and have shown promising results [37][26]. Unfortunately, finding the low-rank decomposition of a tensor (order ≥ 3) is an NP-hard problem and even finding an optimal low-rank approximation is ill-posed [9]. This is why iterative methods are commonly used that minimize the reconstruction error under some L_p norm. Variational Bayesian Matrix Factorization (VBMF) [25] is another alternative that provides a global analytic solution for the rank of a matrix. Practically, this can then be used by unrolling the 4-way convolutional weights using a process known as matricization.

We apply the methodology proposed by Kim *et al.* [18] for an efficient Tucker decomposition implementation (see figure 4) and training pipeline. This is applied to the original HardNet trained weights by using VBMF for rank selection and individually decomposing the weights for each layer. The weights are factorized across the input and output channel dimensions using higher order orthogonal iteration [30] with Tensorly [19]. These factored weights are initialized using the uncompressed network and then fine-tune trained to restore the overall accuracy. We chose not to decompose across the spatial dimensions as the kernels have relatively small receptive field sizes, although this could be considered to achieve marginally higher compression.

3 Model Compression

In this section we introduce our new CDP layer and a linear bottleneck for the depthwise separable layers.

3.1 Convolution-Depthwise-Pointwise (CDP)

Our proposed approach can be motivated by significantly different weight variance for different channels. Figure 5 shows some convolutional weight slices of the pre-trained HardNet++. This weight structure demonstrates a significantly higher variance in weight values across only a subset of the input channels. In fact, we observe a clear and consistent cut-off point (offset) across each layer. We can expect that the higher variance columns (which index a given input channel) are more important and thus will benefit from full depth-wise connectivity, while the low variance can be approximated and accelerated via depthwise and pointwise convolutions. This consistent layer-wise structure across all channel slices motivates our proposed approach.

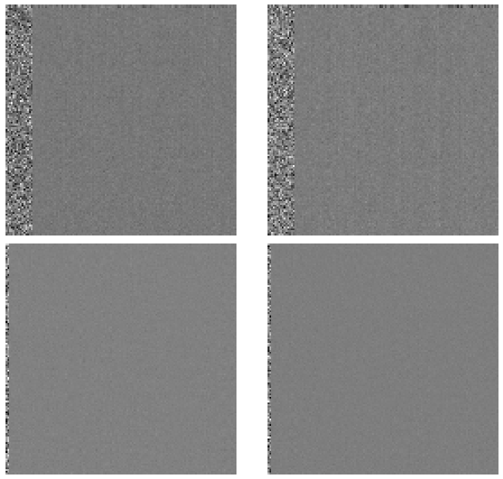

Figure 5: Convolutional weight slices $\mathcal{T}[:,:,w_i,h_j]$ from the pre-trained HardNet++ model. Where i,j is a given spatial coordinate in the receptive field. The top two slices are from layer 6, whereas the bottom two slices are from layer 7. Note that the columns correspond to a given input channel index and the weights are scaled to be in range $[0,255]$ - gray pixels are indicative of weight values near zero.

Based on these observations we propose an approach to combine a depthwise separable layer with a standard convolutional layer to provide some level of full dense spatial and channel connectivity for a subset of the input channels. This is shown in figure 1(c), where the first few channels are reserved for a standard convolution, while depthwise kernels are used for the rest. These output features maps are then concatenated and aggregated using a pointwise convolution. We argue that, although the CDP layer will have more parameters than the typical depthwise separable layers, the dense connectivity will ensure that the high-end accuracy is maintained. We define the number of input channels for the standard convolution to be the offset parameter α. This parameter provides a means of interpolating between a normal convolution layer (albeit with a redundant pointwise convolution), where α is zero, and a fully depthwise separable layer, where α is equal to the number of input channels.

The input feature maps can be represented as a 3-way tensor, $\mathcal{X} \in \mathbb{R}^{W \times H \times C}$, where C indicates the number of feature maps and W, H are the spatial dimensions. We use $\mathcal{X}_{i:j}$ to indicate feature maps indexed from i (inclusive) through to j (non-inclusive). This notation is used to define the operation of the CDP layer proposed. Let $\alpha \in [0, C]$ indicate the depthwise offset for compressing the convolutional layer and $\mathcal{T}^1, \mathcal{T}^2, \mathcal{T}^3$ correspond to the convolution, depthwise, and pointwise kernels respectively.

$$y = (\mathcal{X}_{0:\alpha} *_s \mathcal{T}^1) \oplus (\mathcal{X}_{\alpha:C} *_d \mathcal{T}^2) \quad (7)$$

$$\mathcal{Y} = y *_s \mathcal{T}^3 \quad (8)$$

Where $*_s, *_d$ are the standard convolution and depthwise convolution operators respectively and $\oplus$ is the concatenation of tensors along the depth axis.

The total number of weights for stage (1) is given by $K^2 \cdot \alpha \cdot N + K^2 \cdot (C - \alpha)$, where K^2 is the receptive field size and N is the number of kernels used for the standard convolution block. The output from both these blocks are then concatenated along the depth-axis and followed by a pointwise convolution with O kernels. Both α and N can be adjusted to control the overall compression however, for simplicity, we use $N = O$ throughout. On this basis, compression and acceleration of the overall layer is achieved if $\alpha < C - \frac{N}{K^2 - 1}$ (see Appendix). This result indicates that the CDP layer replacement is more effective on expansion layers where $N > C$.

3.2 Pointwise linear bottleneck

We explore an approach that attempts to combine both tucker decomposition and the depthwise separable layers. This method was motivated by the fact that the pointwise kernels contribute far more significantly to the total computation and the number of parameters than the depthwise kernels. We choose to replace the pointwise convolution in the depthwise separable layer with a bottleneck, where the size of this bottleneck is determined by Tucker decomposition with VBMF. Specifically, the pointwise kernel is decomposed across the input and output channel modes and the largest rank is used as the depth for the bottleneck. Note that the core tensor is not used since the pointwise kernel has unit spatial dimensions and the maximum rank is chosen to ensure restorability of the models accuracy. In this case, model compression and acceleration is only achieved if the depth of the intermediate feature map R is sufficiently small s.t. $CN > CR + RN$. This is unlikely to be met if the channel dimensions are small or the estimated rank is large. This methodology deviates from the Tucker decomposition [18] by the fact that the core tensor is not used. Instead, the pointwise convolution is simply replaced with a direct projection to and from a low-dimensional subspace defined by the largest approximated Tucker decomposition rank - $R = \max(R_1, R_2)$

4 Experimental results

The performance of the descriptor models are evaluated using the HPatches benchmark [3]. We focus on the matching and retrieval tasks as these were found to be more challenging and useful for practical applications [3] unlike the validation task. The HardNet model variants were all implemented in PyTorch [28] and trained with the HardNet loss functions [24]. Note that the HardNet++ weights were trained on the *Liberty*, *Yosemite*, and *Notredame* datasets [7], whereas all the models proposed in this paper are trained solely on the *Liberty* dataset from random initialisation. We use the code provided by the original HardNet paper for training and evaluation[1].

4.1 Dataset and performance metrics

Both the computational cost and the model size were addressed as these are common constraints considered for deployment on devices with limited resources. The model size corresponds to the number of parameters across all the convolutional layers, whereas the computational complexity, measured in the number of floating point operations (FLOPs), considers the cost of the convolutional and activation layers - note that this is not the number of operations *per second*, which is a common metric for hardware performance, but instead the number of floating point multiplications required to compute the output of a given layer. In most cases the cost of the element-wise non-linearity is negligible and the batch normalisation layers are typically fused with the previous layer. The computational complexity puts a theoretical bound on the minimum attainable latency with an efficient GPU implementation. Although the majority of the weights are located in the final layers of the network, which is attributed to the increased depth of the feature maps, the floating point operations are concentrated in the layers where the feature maps have large spatial dimensions. This makes it difficult to balance the performance requirements and so we instead prioritize selectively factorising layers to demonstrate the either ends of the spectrum, where maximum compression is achieved or very efficient computation.

The model size is described through a compression ratio $uncompressed/compressed$ i.e., the total number of parameters in the original network to the total number of parameters in the compressed network. The size of the network is critical for applications that have limited memory and so plays a shared part in the attainable latency on constrained devices, along with the number of computations measured by FLOPs.

4.2 Compressed L2Net performance

We first report the baseline results for state of the art descriptors and then compare the proposed accelerations in terms of network compression ratio, mAP and computational cost (FLOPs). Table 1 (top) compares the number of parameters and HPatches results for three descriptors frequently used in the literature. L2Net [34] and TFeat-M* [4] are shallow CNN architectures and SIFT [22] is a handcrafted descriptor with

[1]https://github.com/DagnyT/hardnet

square root normalisation [2]. The SIFT consists of two convolutions to obtain image gradients which is equivalent to two 5x5 hardcoded kernels, thus 50 parameters.

Depthwise separable architecture. For all the expansion layers we use a width multiplier of 2, as opposed to performing the expansion through the pointwise convolution. We also do not consider the decomposition of the first layer as the number of input channels is 1. Table 1 (bottom) shows the performance and accuracy metrics for using these depthwise separable convolutions as replacements for the standard convolutional layers in the L2Net architecture. Most of the parameters in the network reside in the last convolutional layer (layer 7) and so converting this layer to a depthwise separable convolution dramatically reduces the model size $\sim 4.3\times$. The interesting observation is that doing so has very little effect on the accuracy across both benchmarks. Since the input to the last layer has the same spatial dimensions as the receptive field of the kernel, this layer merely acts as a feature aggregation and projection to the 1×128 descriptor vector. This explains why the network does not need a strong joint spatial and channel-wise connectivity in this last layer to maintain the high-end accuracy. Replacing earlier layers then leads to a small and nearly linear reduction in accuracy along with the number of parameters. In contrast, the majority of the computation is spent in the earlier layers of the network, which is why there is little speed-up by factorising solely the last 3 layers.

Architecture	# Parameters	Image Matching	Patch Retrieval
L2Net	1,334,560	**38.8**	**59.0**
SIFT	50	25.7	42.7
TFeat-M*	599,808	28.7	52.0

Architecture	Compression ratio	Image Matching	Patch Retrieval	Operations
L2Net (HardNet)	$1\times$	**51.1**	**70.5**	35.7M
DepthSep{7}	$4.3\times$	50.1	69.4	34.6M
DepthSep{6-7}	$7.39\times$	47.1	67.5	26.3M
DepthSep{5-7}	$10.72\times$	46.4	67.0	25.4M
DepthSep{2-7}	$18.91\times$	44.5	66.0	**5.6M**

Table 1: The effect of using depthwise separable convolutions for different number of layers in the L2Net architecture trained with HardNet approach. The number(s) in the braces indicates the layers replaced. Image matching and patch retrieval accuracy is evaluated with mean Average Precision (mAP), while the number of operations is in MFLOPs.

A non-linear activation is typically placed in-between the depthwise and pointwise convolutions, however, we observed that this degraded the overall accuracy and so was omitted from the network. This has some obvious implications by considering the linearity of the convolution operation. Specifically, the depthwise and pointwise convolutions can be combined into a single convolution:

$$\mathcal{Y}_{h,w,n} = \sum_{i=1}^{C} (\sum_{k_1,k_2=1}^{K} \mathcal{X}_{h',w',i} \cdot \mathcal{D}_{k_1,k_2,i}) \cdot \mathcal{K}_{i,n} \tag{9}$$

$$= \sum_{i=1}^{C} \sum_{k_1,k_2=1}^{K} \mathcal{X}_{h',w',i} \cdot (\mathcal{K}_{i,n} \cdot \mathcal{D}_{k_1,k_2,i}) \tag{10}$$

$$= \sum_{i=1}^{C} \sum_{k_1,k_2=1}^{K} \mathcal{X}_{h',w',i} \cdot \mathcal{T}_{k_1,k_2,n} \tag{11}$$

Which is a standard 2D convolution with $\mathcal{T}$, that is derived by contracting the original depthwise filters and pointwise filters over the shared input-channel dimension. Performing this contraction a-priori will incur the same computational overhead as the standard convolution and so the operations are instead performed in sequence (as is defined by the original depthwise separable layer).

Tucker decomposition. Networks with such decomposed layers have a different connectivity pattern, which may be more effective in the early layers for designing a more computational efficient model variant. This layer projects the input feature map to a lower dimensional subspace, where the core tensor is applied [18]. We observe that, unlike with depthwise separable convolutions, factorizing all but the last layer i.e., Tucker{2-6}, significantly reduces the number of operations (by over $3\times$) with minimal accuracy degradation (see table 2). However, applying the Tucker decomposition to the final layer will in fact significantly degrade the accuracy across both tasks through the significant compression across the depthwise dimension. No batch normalisation or non-linearities were placed in between the stages shown in figure 4 based on an observed drop in accuracy.

Architecture	Compression ratio	Image Matching	Patch Retrieval	Operations
L2Net (HardNet)	$1\times$	**51.1**	**70.5**	35.7M
Tucker{2-6}	$1.21\times$	50.7	70.1	10.3M
Tucker{7}	$3.11\times$	29.7	50.1	34.8M
Tucker{2-7}	$\mathbf{6.81\times}$	21.8	42.2	**9.4M**

Table 2: The effect of using Tucker decomposition in different layers of the original network. The rank selection is done through VBMF.

Pointwise linear bottleneck. As previously demonstrated, replacing the last convolutional layer (layer 7) with a depthwise separable layer leads to a strong accuracy vs model-size trade-off and, in contrast, applying the Tucker decomposition to layers 2-6 provided a good accuracy vs computation trade-off. However, quite surprisingly, it appears that naively attempting to combine the benefits of both these networks leads to a significant degradation in the overall accuracy as shown in table 3. The alternative consideration is to replace the pointwise kernel with a linear bottleneck layer, as described in section 3.2. This proved to be a more effective method for combining the benefits of both the depthwise separable and Tucker variants. By using this approach, the network provides a more balanced trade-off between the performance metrics, while maintaining high accuracy. Layer 5 consists of the following embedding $64 \rightarrow 33 \rightarrow 128$, while layer 6 projects to a larger bottleneck $128 \rightarrow 43 \rightarrow 128$. The VBMF evaluated rank for layer 7 was too high for any compression or acceleration and so we retained the original depthwise separable layer. The most notable observation with these results is that replacing the pointwise convolution with a linear bottleneck marginally improves the accuracy across both tasks in comparison to the standalone depthwise separable layers.

Architecture	Compression ratio	Image Matching	Patch Retrieval	Operations
L2Net (HardNet)	$1\times$	**51.1**	**70.5**	35.7M
DepthSep{7} + Tucker{2-6}	**20.56**$\times$	27.4	47.3	**8.5M**
DepthSep{7} + TDW{5-6}	$12.01\times$	47.0	67.2	25.0M

Table 3: Combined approach of Tucker decomposition and depthwise separable layers. TDW layers describe the combined approach, where the pointwise kernel is replaced with a bottleneck layer with a width defined by the estimated VBMF rank.

CDP Layers. The results from table 4 demonstrate how enabling a subset of the channels to utilise full dense connectivity allows for the model to reach the high-end accuracy, while still achieving the favourable compression and acceleration from depthwise separable layers. The first group of rows considers naively using a fixed *offset* for each layer, the second group takes into account the expansion of every odd layer by doubling the *offset* on each of these layers, and finally the bottom row uses the *offsets* as defined from the pre-trained HardNet++ weights (see figure 5). We observe that, as long as there is at least some full dense connectivity (attributed to the standard convolution), the architecture is able to achieve the top-end accuracy. This top-end accuracy was not attainable for the usage of just depthwise separable layers, Tucker decomposition, or even the use of a pointwise linear bottleneck. In fact, the CDP variants are able to achieve the best balance between the number of parameters and computation, while also demonstrated very little drop in accuracy ($< 1\%$).

4.3 SuperPoint

We explore how our CDP layer can be applied to other architectures and datasets for significant performance improvement. For this we consider the SuperPoint [10] model, which leverages a VGG [41] backbone to jointly predict the interest points and descriptors for matching. The results for the descriptor task are compared on the *HPatches* dataset across illumination and viewpoint changes.

We use the original SuperPoint code[2] and introduce our modifications for all the model variants. To ensure consistency, we follow the original training methodology for all the evaluations, which includes the same homographic adaptions of training images. The results for the detector evaluation can be seen in table 5, where the comparative results for the classical detectors can be seen at the top, while the SuperPoint variants can be seen at the bottom. The SuperPoint model is significantly more computationally demanding than the L2Net model as the input image has much larger spatial dimensions, which is attributed to the joint detection and descriptor evaluation. However, it is worth noting that the forward pass for the L2Net model must be performed for every interest patch, typically 1000s per image, whereas the number of operations shown in all the tables is for a single forward pass of batch size 1. We evaluate the results on a relatively large input image ($240 \times 320 \times 3$) however, this may be inpractical for real-time mobile applications. Scaling the input image will linearly scale the number of operations required throughout the network and would need to be considered for applications where the resource constraints are known.

Applying the CDP layer to the entire network maintains the final repeatability of the detector (withing 2.d.p of mAP), while still achieving significant compression and acceleration.

[2]https://github.com/rpautrat/SuperPoint

5 Conclusions

In this paper we demonstrate the accuracy/performance tradeoffs of applying various factorisation and networks compression methods on the CNN models for local feature extraction. We have proposed a novel Convolution-Depthwise-Pointwise (CDP) layer that provides a means of efficiently factorizing a network, while maintaining dense connectivity in high variance layers leading to the top-end descriptor matching accuracy. We demonstrate the generalisability of this idea onto large architectures, namely the SuperPoint detector with VGG based encoder. By replacing the standard convolutional layers with CDP layers, we are able to maintain the repeatability while offering a $\sim 3\times$ reduction in the number of parameters and a $\sim 3\times$ reduction in computational complexity measured by FLOPs. In the case for the widely used L2Net architecture, we able to achieve an $8\times$ reduction in parameters, $2.6\times$ reduction in FLOPs with less than 1% drop in accuracy over both the image matching and patch retrieval tasks.

Our experimental evaluation in this paper considers the typical parameter settings when balancing both the standard accuracy and performance metrics of a model. We expect that an exhaustive parameter search could provide models with improved efficiency.

Both the SuperPoint and L2Net CDP variants are now shown to have a significantly lower cost in terms of memory and computation for their usage on mobile devices.

References

[1] P. Alcantarilla, J. Nuevo, and A. Bartoli. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. *BMVC*, 2013.

[2] R. Arandjelovic. Three Things Everyone Should Know to Improve Object Retrieval. *CVPR*, 2012.

[3] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. HPatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *CVPR*, 2017.

Layer offsets						Image Matching mAP [%]	Patch Retrieval mAP [%]	Compression ratio	Operations (MFLOPs)
#2	**#3**	**#4**	**#5**	**#6**	**#7**				
Original						**51.1**	**70.5**	1×	32.35
2	2	2	2	2	2	48.6	68.6	**9.50×**	10.65
5	5	5	5	5	5	50.3	70.4	7.66×	12.43
10	10	10	10	10	10	50.0	70.0	5.79×	15.43
15	15	15	15	15	15	50.4	70.3	4.65×	18.42
2	4	4	8	8	16	50.0	70.0	5.01×	11.76
4	8	8	16	16	32	50.1	70.1	3.21×	14.07
4	8	8	16	16	2	49.9	69.9	7.61×	13.83

Table 4: Applying different depthwise offsets for the L2Net architecture with CDP layers. The results are evaluated on the HPatches benchmark and averaged over the Easy, Hard, and Tough distributions.

Architecture	Illumination changes	Viewpoint changes
FAST	57.6	41.5
Harris	63.0	48.1

Layer offsets									Illumination changes	Viewpoint changes	Compression ratio	Operations (GFLOPs)	
#2	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**				Backbone	Head
Original									67.3	39.5	1×	5.81	0.37
2	2	2	2	2	2	2	2	2	67.0	38.8	**3.21×**	**1.61**	**0.15**
5	5	5	5	5	5	5	5	5	**67.3**	**40.5**	2.97×	1.83	0.16
2	4	4	8	8	16	16	16	16	66.8	39.6	2.58×	1.74	0.18

Table 5: Repeatability for the classical detectors and the SuperPoint variants on the *HPatches* dataset. All the models are evaluated on 1000 interest points. The first 8 layers correspond to the shared VGG net architecture, whereas the last 2 layers form the detector head. The FLOP calculation assumes an input image of size $240 \times 320 \times 3$.

[4] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. *BMVC*, 2017.

[5] A. Barroso Laguna, E. Riba, D. Ponsa, and K. Mikolajczyk. Key.Net: Keypoint Detection by a Hybrid of Handcrafted and Learned CNN Filters. page arXiv:1904.00889, 2019.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Vangool. Speeded-Up Robust Features (SURF). *CVIU*, 2008.

[7] M. Brown and D. G. Lowe. Automatic Panoramic Image Stitching Using Invariant Features. *IJCV*, 2007.

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Binary Robust Independent Elementary Features. *ECCV*, 2010.

[9] V. de Silva and L.-H. Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM*, 2006.

[10] D. Detone, T. Malisiewicz, and A. Rabinovich. SuperPoint: Self-supervised interest point detection and description. *CVPR*, 2018.

[11] M. H. Fox, K. Kim, and D. Ehrenkrantz. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *CVPR*, 2018.

[12] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. Technical report, 1993.

[13] K. He, Y. Lu, and S. Sclaroff. Local descriptors optimized for average precision. *CVPR*, 2018.

[14] K. He, X. Zhang, S. Ren, and J. Sun. ResNet - Deep Residual Learning for Image Recognition. *CVPR*, 2015.

[15] F. L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 2015.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, and M. Andreetto. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CVPR*, 2017.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. *BMVC*, 2014.

[18] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression Of Deep Convolutional Neural Networks For Fast And Low Power Mobile Applications. *ICLR*, 2016.

[19] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic. TensorLy: Tensor Learning in Python. *JMLR*, 2016.

[20] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. *ICCV*, 2011.

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[22] D. G. Lowe. SIFT - Distinctive Image Features from Scale-Invariant Keypoints. *ICCV*, 2004.

[23] Z. Luo, T. Shen, L. Zhou, S. Zhu, R. Zhang, Y. Yao, T. Fang, and L. Quan. GeoDesc: Learning local descriptors by integrating geometry constraints. *CVPR*, 2018.

[24] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. *NeurIPS*, 2017.

[25] S. Nakajima, M. Sugiyama, S. D. Babacan, and R. Tomioka. Global analytic solution of fully-observed variational Bayesian matrix factorization. *JMLR*, 2013.

[26] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *NeurIPS*, 2015.

[27] Y. Ono, E. Trulls, P. Fua, and K. Yi. Lf-net: Learning local features from images. *NeurIPS*, 2018.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. *NeurIPS*, 2017.

[29] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. In *ECCV*, 2006.

[30] B. N. Sheehan and Y. Saad. Higher Order Orthogonal Iteration of Tensors (HOOI) and its Relation to PCA and GLRAM. *SIAM*, 2007.

[31] L. Sifre and S. Mallat. PhD Thesis, Ecole Polytechnique, CMAP Rigid-Motion Scattering For Image Classification. 2014.

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *CoRR*, 2015.
[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. GoogLeNet/Inception - Going deeper with convolutions. In *CVPR*, 2015.
[34] Y. Tian, B. Fan, and F. Wu. L2-Net: Deep Learning of Discriminative Patch Descriptor in Euclidean Space. *CVPR*, 2017.
[35] Y. Tian, X. Yu, B. Fan, F. Wu, H. Heijnen, and V. Balntas. Sosnet: Second order similarity regularization for local descriptor learning. *CVPR*, 2019.
[36] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 1966.
[37] W. Wang, B. Eriksson, and W. Wang. Wide Compression : Tensor Ring Nets. *CVPR*, 2018.
[38] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. *NeurIPS*, 2016.
[39] Yann Lecun. Optimal Brain Damage. *NeurIPS*, 1990.
[40] X. Zhang, X. Zhou, and M. Lin. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *CVPR*, 2017.
[41] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE T-PAMI*, 2016.

6 Appendices

6.1 Compression ratio of the CDP layer

The CDP layer replaces the standard convolution operation with a smaller convolution that acts on a subset of the input channel dimensions. A depthwise convolution is then used for the remaining feature maps and a pointwise convolution is used for fusing features from both. With a depthwise offset given by α, the total number of weights is given by the sum of each of the contributing blocks respectively.

$$\begin{aligned} &K^2 \times \alpha \times N \\ &+ K^2 \times (C - \alpha) \\ &+ 1 \times 1 \times (N + (C - \alpha)) \times N \end{aligned}$$

Note that we make the assumptions that there are N convolutional kernels and N pointwise kernels, however, further compression could be achieved by reducing the number of standard convolutional kernels. The overall compression is achieved if α satisfies the following inequality:

$$K^2 \cdot \alpha \cdot N + K^2 \cdot (C - \alpha) + (N + (C - \alpha)) \cdot N < K^2 \cdot C \cdot N$$

Solving for α, this can then be reduced to

$$\begin{aligned} \alpha &< \frac{K^2 \cdot C \cdot N - K^2 \cdot C - N^2 - C \cdot N}{K^2 \cdot N - K^2 - N} \\ &= \frac{C \cdot (K^2 \cdot N - K^2 - N) - N^2}{K^2 \cdot N - K^2 - N} \\ &= C - \frac{N^2}{K^2 \cdot N - K^2 - N} \\ &= C - \frac{N^2}{K^2 \cdot (N-1) - N} \end{aligned}$$

By assuming $N \gg 1$, the bound on α is further restricted but significantly simplified.

$$\alpha < C - \frac{N}{K^2 - 1}$$

6.2 Computational cost of the CDP layer

The cost of applying the element-wise ReLU operation to the intermediate feature map of both the depthwise and standard convolution is given by:

$$W \times H \times (N + (C - \alpha))$$

Thus, the overall computational cost of the CDP layer is as follows:

$$WH \cdot (K^2 \alpha N + K^2 (C - \alpha) + (N + (C - \alpha)) + (N + (C - \alpha))N)$$

The component parts are for the standard convolution, depthwise convolution, element-wise ReLU and pointwise convolution respectively. From this, the computational speedup can be derived:

$$\frac{C \cdot N}{\alpha N + C - \alpha} + \frac{K^2 \cdot C}{1 + N + C - \alpha} + \frac{K^2 \cdot C \cdot N}{C - \alpha}$$

Similarly for the compression, we can also derive an inequality for the requirement on α for achieving a reduction in the computational cost.

$$\begin{aligned} \alpha &< \frac{K^2 CN - K^2 C - N - C - N^2 - NC}{K^2 N - K^2 - 1 - N} \\ &= C + \frac{N \cdot (N+1)}{K^2 \cdot (1 - N) + N + 1} \end{aligned}$$

Using the same assumption that $N \gg 1$ further simplifies this inequality:

$$\begin{aligned} \alpha &< C + \frac{N^2}{N - K^2 N} \\ &= C - \frac{N}{K^2 - 1} \end{aligned}$$

Which is the same condition imposed on the compression of parameters. This is a result of the imposed assumption that $N \gg 1$, which effectively ignores the cost of the ReLU operation. This is consistent in practise, where the cost of the ReLU operation is often not significant.