OnionNet: Sharing Features in Cascaded Deep Classifiers

Martin Simonovsky
martin.simonovsky@enpc.fr
Nikos Komodakis
nikos.komodakis@enpc.fr

Imagine Lab
Université Paris Est / École des Ponts
Paris, France

Abstract

The focus of our work is speeding up evaluation of deep neural networks in retrieval scenarios, where conventional architectures may spend too much time on negative examples. We propose to replace a monolithic network with our novel cascade of feature-sharing deep classifiers, called OnionNet, where subsequent stages may add both new layers as well as new feature channels to the previous ones. Importantly, intermediate feature maps are shared among classifiers, preventing them from the necessity of being recomputed. To accomplish this, the model is trained end-to-end in a principled way under a joint loss. We validate our approach in theory and on a synthetic benchmark. As a result demonstrated in three applications (patch matching, object detection, and image retrieval), our cascade can operate significantly faster than both monolithic networks and traditional cascades without sharing at the cost of marginal decrease in precision.

1 Introduction

The last several years have seen deep neural networks (DNNs) bringing tremendous rise in performance to variety of recognition tasks. However, this often comes at a price of high computational cost at test time, the reduction of which has recently become a hot topic in deep learning [17, 29, 40]. Particularly in retrieval scenarios, large amount of computational time may be spent on negative examples of varying difficulty.

A popular remedy is to set up a cascade of multiple classifiers of increasing strength, called stages [35]. Recently, a pair of independent DNNs was used in a cascade [2, 21, 41]. Also the Region proposal network of Faster R-CNN [29] can be essentially seen as the first stage in a two-stage cascade. While in the former case both networks receive the raw input and build up their higher-level representation individually, in the latter case the stages are finetuned to share their first five convolutional layers. As these are the most expensive ones to compute [15], it is questionable whether such powerful features are always necessary.

Our observation is that these are the extreme cases of sharing. If the intermediate representation is not reused, a representation presumably at least as powerful as before has to be rebuilt in the following stage and the running time for positive examples suffers. On the other hand, making the first stage use the representation of the last stage may lead to losing time on easy negatives.

Pages 79.1-79.13
DOI: https://dx.doi.org/10.5244/C.30.79

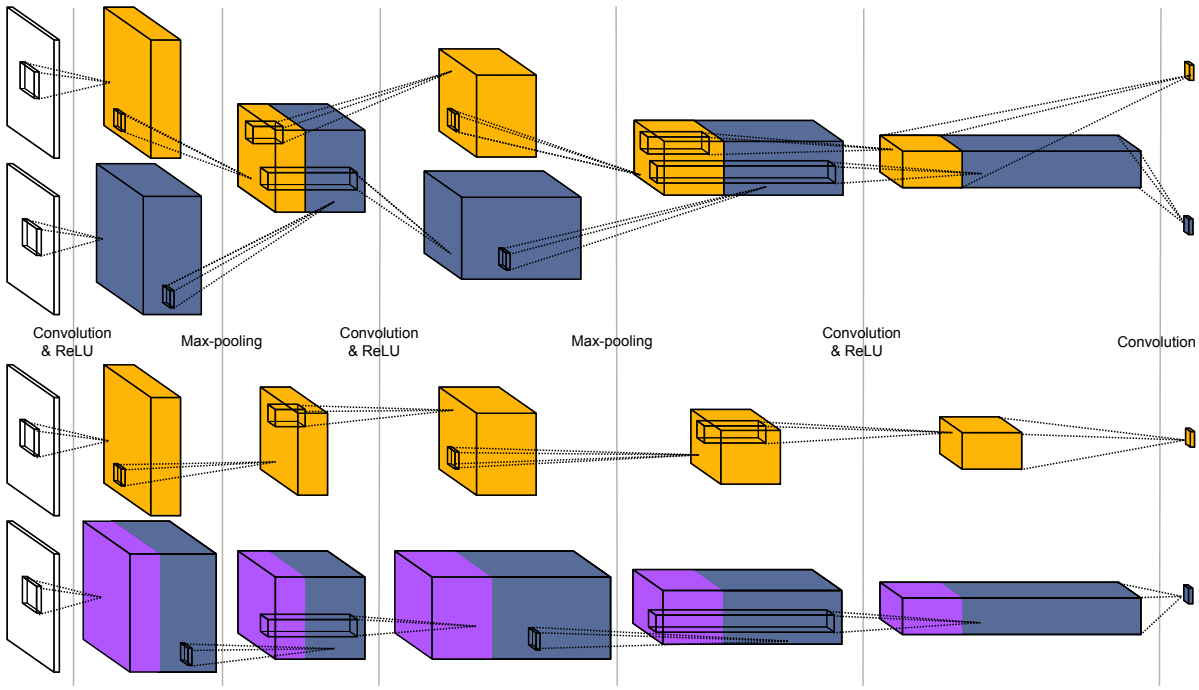

Figure 1: Feature map sharing. *Top*: Two-stage OnionNet. *Bottom*: A corresponding non-sharing cascade. In OnionNet, the first stage (S1, orange) shares its intermediate feature maps (visualized as cubes) with the second stage (S2, blue). Without sharing the stages are independent and S2 has to be evaluated fully, recomputing certain features (purple).

We address this by proposing OnionNet, a novel architecture where the next stage extends the feature map set of the previous stage, preventing repeated computation. Crucially, the architecture is flexible: the next stage may add both new layers as well as new feature channels, while reusing the previous ones at the same time. Thus, our stages do not have to be of increasing depth only, even classifiers of the same depth but increasing width are still able to share their features. To accomplish this, the model is trained end-to-end in a principled way under a joint loss.

OnionNet is demonstrated in three important tasks: patch matching, proposal-based object detection, and image retrieval. We achieve substantial speed-up compared to non-cascaded baselines as well as non-sharing cascades, with only a marginal loss in precision.

As our main contributions we show that cascaded DNN may offer significant computational benefits compared to monolithic architectures, propose a novel cascaded architecture that promotes feature sharing leading to additional computational advantages, and provide a systematic study that sheds further light into the time cost behavior of cascaded architectures.

2 Related Work

Cascades and Sharing. Whereas in the pioneering work of Viola and Jones [35] stages are distinct and essentially trained with hard negative mining, the soft cascades of Bourdev and Brandt [3] are trained as a single boosted classifier where each weak learner has a cumulative score rejection threshold. We are motivated by the general idea of stages building successively on each other and realize it in the context of DNNs. Zehnder *et al.* [39] share stages among several class-specific cascades for multi-class detection, but the stages itself are independent. In deep learning, the sharing idea of Faster R-CNN [29] comes probably the closest to our method. However, their training is less principled than ours, using a '4-step training

algorithm to learn shared features via alternating optimization'. Moreover, our architecture is more flexible, as subsequent stages can also add new feature channels besides new layers.

We also note that the term 'cascades' is overloaded in the literature. Several authors [12, 33, 34] speak of cascades to describe a sequence of stages, evaluated as whole, where one stage receives the output of another and further refines it. Our cascades aim for early rejection of negatives.

Conditional Execution. Our approach is also related to conditional evaluation of networks. In the hierarchical classification with HD-CNN [36], class group specialist networks are executed based on the prediction of a group classifier, all sharing early layers. Unlike our approach, HD-CNN aims for precision rather than speed. Dynamic Capacity Network [1] uses an entropy-based attention mechanism to apply a more expensive network to salinient parts of the input image for better prediction. Our work processes images as a whole and concentrates on feature sharing instead.

Model Compression. The research on speeding up the evaluation of DNNs is related in general, especially the works exploring redundancy in networks. Knowledge distillation [17, 30] aims to compress models in a student-teacher framework, whereas matrix factorization methods [11, 19, 40] replace weight matrices by their low-rank approximations. Computational efficiency can be also incorporated from the beginning by imposing *e.g.* a special filter structure [7] or sparse filter connectivity [18]. In a sense, we also exploit redundancy present in our baselines, assuming it is possible to separate a certain amount of layers/channels into an individual stage, which still performs reasonably well on the same (sub)task. However, our motivation is different, we train our cascade concurrently from scratch with the aim to use the full, combined network as the last stage as well.

3 Method

Our model is a cascade of feed-forward DNNs, called stages, evaluated sequentially at test time. The aim of the cascade is to confidently discriminate an input example as early in the classifier pipeline as possible, saving running time. To deal with gradually more complicated examples, the later stages should be more refined and operate on a higher level of abstraction.

Motivation. Multiple networks of different sizes trained on the same dataset and for the same or a similar objective raise the question whether their learned features have something in common. Li *et al.* [25] confirm this for the case of different initializations of the same network. Our major assumption is that the feature maps at a particular layer computed by a smaller network can be approximately subsumed by the feature maps of a larger network. Thus, the larger network can be seen as an envelope around the smaller network, adding new feature channels or layers and (partially) reusing the features of the smaller network. Specifically, each convolutional layer of the larger network receives the respective feature maps from all smaller networks as an additional input. The key observation is that these are shared and don't have to be recomputed. Our cascade, coined OnionNet, can be pictured as an onion, each next stage wrapping the previous.

Depth vs. Width. A natural way of constructing such a cascade might be to gradually increase the depth only. This is principally similar to training a deeply supervised network [23] and proceeding to deeper layers at test time until an associated 'local companion output' rejects the example. However, the first layers are the most expensive to compute due to large spatial size [15] while tending to produce weak classifiers due to few non-linearities [32]. Instead, we assume it is likely that early stages of the cascade don't need as many feature

maps as the later ones, which leads us to construct the cascade by gradually increasing the width, possibly in addition to depth. Making a stage thin reduces the burden significantly and permits the cascade to delay fully evaluating expensive lower layers until necessary.

In the rest of the paper, we restrict our scope to a two-stage cascade only. However, our approach can be easily generalized to cascades with more stages.

3.1 Model Description

Our two-stage OnionNet cascade consists of two branches with the same layer organization (Figure 1). Each takes the same input and is terminated by its own output layer. The core idea is that the branches are linked before every convolutional layer, including the final one. The feature maps of the first stage (S1) are used as additional input to the following convolutional layer in the second stage (S2) but not the other way round, creating a one-way dependence. Let n_l^N denote the number of filters in the l-th convolutional layer C_l^N of network N. Then C_l^{S2} receives its combined input of size $n_{l-1}^{S1} + n_{l-1}^{S2}$ from both S2 and S1 and, conversely, the output of the layer immediately preceeding C_l^{S1} (usually a ReLU or a max-pooling layer) is forwarded to both C_l^{S1} and C_l^{S2}.

In applications, OnionNet is designed as a replacement for a large monolithic network $N_{\mathtt{M}}$. A simple way to configure a cascade is to keep the effective number of filters per layer unchanged, i.e. splitting n_l^M filters of C_l^M to n_l^{S1} and n_l^{S2} filters, where $n_l^{S1} + n_l^{S2} = n_l^M$. Although the number of feature maps is preserved, the amount of weights decreases due to missing connections from S2 to S1 by $s_l^2 n_{l-1}^{S2} n_l^{S1}$, where s_l denotes the size of filters (common to all $N_{\mathtt{M}}$, S1, and S2). This has the same, albeit less severe effect on both speed and accuracy as the so-called filter groups, which arise when splitting layers among multiple GPUs [22] or by imposing structure-induced regularization [18].

3.2 Training

Each stage is assigned its own loss function L^N, evaluated on the output layer. Whereas L^{S2} is application dependent, L^{S1} is the standard cross-entropy loss over set of S1-classes $\mathcal{K}$. OnionNet is trained jointly as a single model under the combined loss $L = \alpha L^{S1} + (1-\alpha) L^{S2}$, where $\alpha \in (0,1)$ is a fixed hyperparameter, each stage having access to the full training set. Due to feature map sharing between the branches, the weights of S1 (except for the last layer) receive backpropagation updates from both L^{S1} and L^{S2}, while the weights of S2 are trained under L^{S2} only. Therefore, the major benefit of joint training is that the cascade learns the allocation of features between the networks guided by the ratio of individual losses. In our initial experiments with stage-wise independent training, we observed decreased accuracy of S2 and an increased need for technical tweaks for it to properly converge.

3.3 Testing

Thresholds. We fix desired true positive rates (TPRs, recall) for S1-classes of user interest $\mathcal{U} \subset \mathcal{K}$, as we care to precisely control the final accuracy rather than speed (false positive rate, FPR). In order to choose such thresholds in a principled manner, ROC curve is computed for each S1-class based on the score statistics over the complete training set. A test example passes S1 if it scores above any of $|\mathcal{U}|$ predefined thresholds, otherwise it is rejected

Sparse Batches. Not all examples in a test batch may pass the first stage, leaving an irregular pattern of holes for S2. Unfortunately, these cannot be easily skipped as no current GPU backend can work with irregularly strided memory blocks. Thus, the batch as well as

$\texttt{P}_\texttt{M}$ [37]	C4(96), 3×C3(96), 3×C3(192), C2(1)
P	C4(32/64), 3×C3(32/64), 3×C3(64/128), C2(2/1)
$\texttt{D}_\texttt{M}$ [22]	C11(96), C5(256), 2×C3(384), C3(256), C6(4096), C1(4096), C1(21)
D	C11(96), C5(256), 2×C3(384), C3(256), C6(512/3584), C1(512/3584), C1(2/21)
$\texttt{R}_\texttt{M}$ [5]	C11(64), C5(256), 3×C3(256), C6(4096), C1(4096), C1(4096)
R	C11(8/56), C5(32/224), 3×C3(32/224), C6(256/3840), C1(256/3840), C1(7/1000)

Table 1: Configuration of the monolithic baselines ($\texttt{P}_\texttt{M}$ for patch comparison, $\texttt{D}_\texttt{M}$ for object detection, and $\texttt{R}_\texttt{M}$ for image retrieval) and their best OnionNet models (P, D, R). Only parametric layers are listed for clarity, other layers and parameters are consistent with the paper having introduced the baseline. Fully connected layers are implemented as convolutions. Cs(n) denotes a convolutional layer with n output filters of spatial size $s \times s$. In OnionNets, Cs(n^{S1}/n^{S2}) denotes a pair of convolutional layers: Cs(n^{S1}) in S1 and Cs(n^{S2}) in S2.

all shared feature maps have to be reshuffled into smaller contiguous blocks and the output of S2 scattered back. Formalized as obtaining a single contiguous subsequence of 1s in a binary vector with the least amount of move operations, this classic problem is solved efficiently in just two passes over the vector.

4 Evaluation

In this section, we evaluate OnionNet in three different applications: descriptor matching, object detection, and image retrieval.

In each application we compare the best-performing OnionNet cascade N to its respective monolithic baseline network $N_\texttt{M}$. The configurations, listed in Table 1, are designed to keep the effective number of filters in S2 as in $N_\texttt{M}$ (Section 3.1). The non-sharing cascade $N_\texttt{NS}$ constitutes our second baseline; its stages do not share features but have the same number of effective filters in each stage as N.

Implementation was done in Torch [9] with cuDNN backend [8] with auto-tuning to use the fastest convolution algorithms. Mean running time over 50 executions on NVIDIA Titan Black is reported with its standard error. We time solely the forward pass, and not *e.g.* any preprocessing or uploading of the batch. In addition, we report $\bar{p}$ as the mean percentage p of examples in a batch passing S1, which is indicative of the strength of the first stage.

4.1 Application: Comparing Patches

DNNs have been applied to comparing patches just recently, achieving state-of-the-art results. While the ultimate goal might be to learn L2 embeddings of deep descriptors [31], comparing descriptor pairs using a matching network [14, 37, 38] and particularly processing patch pairs jointly from the start were shown to be the best-performing solutions so far [37]. However, as there are quadratically many pairs and the joint (sub)network has to be evaluated for each comparison, such architectures seem rather impractical. Fortunately, the expected high number of easy negative pairs makes for a natural application of OnionNet. This holds especially true for feature point matching between images.

Setting. We evaluate on two datasets. The multi-view stereo correspondence dataset (MVSD) of Brown *et al.* [4] is a balanced dataset of grayscale patches. There are three subsets; we train on Notre Dame and test on Liberty and Yosemite, reporting the false positive rate at 95% recall as in [37]. The local descriptor benchmark (LDB) of Mikolajczyk and Schmid [27] consists of 6 images sequences with ground truth homographies. This dataset is

	FPR@95	$\bar{p}$	running time [sec]
$\mathtt{P_M}$	0.0503	100.00%	12.111 ± 0.001
$\mathtt{P_{NS}}$	0.0514	55.44%	9.677 ± 0.005
$\mathtt{P}$	0.0601	54.05%	8.407 ± 0.006

Table 2: Descriptor matching (MVSD). Average over Liberty and Yosemite subsets at TPR=0.99 on S1. FPR@95 is FPR at TPR=0.95 on S2/baseline, $\bar{p}$ is mean percentage of examples passing S1, running time is normalized per 100K examples.

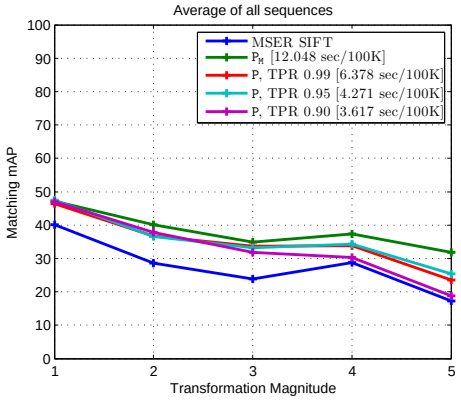

Figure 2: Descriptor matching (LDB).

expected to have an unbalanced, realistic proportion of positives and negatives. We use the framework of [24] for evaluation, regions of interest being extracted using MSER detector.

To fully demonstrate our advantage, we experiment with "2ch-deep" model $\mathtt{P_M}$ from Zagoruyko and Komodakis [37]. Note that our method could also be applied to matching networks in the same way. All models were trained from scratch for 256 epochs with L^{S2} being binary hinge loss and $\alpha = 0.5$. ASGD with learning rate 0.1, weight decay 0.0005, and momentum 0.9 was used to train the models with batch size 128 and random flipping.

Results. Table 2 lists the results on MVSD with TPR of S1 set to 0.99[1]. OnionNet outperforms its baselines in terms of running time (by 31% for $\mathtt{P_M}$ and 13% for $\mathtt{P_{NS}}$). The better mAP but worse running time of $\mathtt{P_{NS}}$ w.r.t. $\mathtt{P}$ is justified in Section 5.1. Figure 2 plots mAP and running times on LDB averaged over all types of transformations in the dataset. The evaluation of LDB does not constrain us from choosing a larger set of TPRs: 0.99, 0.95, and 0.90. The plot shows that under a more realistic imbalance of positive and negative examples we can archive considerable speedup (up to 2.8x) with limited decrease of precision, which starts to show up mostly under higher transformation magnitude. This is likely caused by discarding positives difficult due their extreme deformation, which places them near the decision boundary.

4.2 Application: Proposal-based Object Detection

The currently dominant paradigm in object detection is to use an algorithm to generate a set of object proposals, which are then verified by a classifier. Object proposal algorithms are typically tuned for high recall and are often class-agnostic, which allows them to be used as an off-the-shelf preprocessing step. This flexibility comes at a price of the classifier having to process many proposals that are of no interest with respect to the task-specific set of classes. For example, Fast R-CNN sifts through thousands of proposals per image [13]. Motivated by this, we propose to construct the classifer as OnionNet so that its first stage serves as a background classifier, leaving the task of identifying the particular classes to the second stage. Note that such a task-specific scoring can be alternatively built into the proposal generator itself, as demonstrated by several very recent works [28, 29].

Setting. We experiment with Fast R-CNN [13] on PASCAL VOC 2007 with precomputed Selective Search proposals available at the author's webpage. The baseline $\mathtt{D_M}$ is their 'small' AlexNet network (our reproduction scores 0.023 mAP less). Our proposal $\mathtt{D}$ is cre-

[1]Setting S1-TPR to 0.98 already made it produce enough false negatives so that the prescribed S2-TPR of 0.95 was never reached. This is mostly due to domain transfer, as the particular threshold for S1 was chosen based on ROC curve computed on the training set of a different subset.

ated by replacing the classifier, composed of 3 fully-connected layers, by OnionNet of the same width, see Table 1. The classifier layers were initialized randomly in all models and the whole networks were finetuned on trainval subset of VOC 2007 for 120k iterations using SGD with $\alpha = 0.5$, L^{S2} as cross-entropy loss, and learning rate 0.001, dropping to 0.0001 after 100k iterations. Bounding box regression was omitted in the implementation [26], which does not affect conclusions from our comparison, though. We report mean average precision (mAP) with S1 TPR fixed at 0.95.

Results. The results are listed in Table 4. OnionNet D is able to achieve 2.9x gain in speed w.r.t. baseline $\mathtt{D}_\mathtt{M}$ under a graceful degradation of 0.018 mAP points, which is better than the 1.72× speed-up attained by SVD of the weight matrices as in [13]. Note that both methods might be combined as they are basically orthogonal. We also marginally outperform $\mathtt{D}_\mathtt{NS}$ in both running time and precision.

4.3 Application: Image Retrieval

We are motivated by retrieval over ephemeral datasets, where an index building stage typical for image retrieval [20] may be too heavy; consider *e.g.* a robot actively searching for a particular object or a user wanting to copy images of only cats from his camera. Instead, on-the-fly retrieval [6] casts such a problem as classification. However, similar to the way human search, the system does not need to precisely label every object it knows unless it is the object being searched for. We demonstrate that designing the classifier as OnionNet can lead to a significant decrease in running time.

Setting. We train and validate on ILSVRC 2012 [10]. Although not perfectly suited for retrieval scenarios due to incomplete annotations [6], we choose it because of its scale and our concentration on quantifying relative performance improvements. We aim for retrieving images of a certain class from the set of 50k validation images, rather than classifying all images. Therefore, as in PASCAL VOC classification task, we report mean average precision (mAP) over 1000 classes instead of accuracy. A test example is considered retrieved if its true class is predicted within the top-5 softmaxed scores. TPR of S1 is fixed at 0.9. S2-classes are partitioned into 7 S1-classes $\mathcal{K}$ by k-means clustering of class-averaged activation.

Experiments are performed with Alexnet-like 'CNN-F' baseline $\mathtt{R}_\mathtt{M}$ from [5]. All models were trained from scratch for 53 epochs with L^{S2} being 1000-way cross-entropy loss and $\alpha = 0.5$. SGD with learning rate 0.01 (reduced to 0.005, 0.001, 0.0005, 0.0001 after 18, 29, 43, 52 epochs), weight decay 0.0005 for 29 epochs, and momentum 0.9 was used to train the models with batch size 128. As a reference, our $\mathtt{R}_\mathtt{M}$ achieves 19.3% top-5 error with 10 crops on the standard ILSVRC classification task. The S1 network of our proposal R is as deep as $\mathtt{R}_\mathtt{M}$ and contains $1/8$, resp. $1/16$ of its convolutional, resp. fully-connected filters, see Table 1.

Results. The results are listed in Table 3. OnionNet is able to cut the running time by 41.5% while giving up only 0.049 mAP points w.r.t. baseline $\mathtt{R}_\mathtt{M}$. It also saves 7% time w.r.t. non-sharing cascade $\mathtt{R}_\mathtt{NS}$, which is a fair result given the relatively low amount of shared feature maps. The better mAP but worse running time of $\mathtt{R}_\mathtt{NS}$ w.r.t. R is justified in Section 5.1.

5 Discussion

In this section we conduct further analysis in order to gain insight into the properties of OnionNet. To that end, we define multiple variants of OnionNet for the image retrieval network $\mathtt{R}_\mathtt{M}$ by varying the width or depth of S1. We extend our notation by superscripts for that: the S1 network of $\mathtt{R}^{\mathtt{W}w}$ has the width of grade w (the greater the wider), the depth d of $\mathtt{R}^{\mathtt{D}d}$

	mAP	$\bar{p}$	running time [sec]
$\mathtt{R_M}$	0.587	100%	49.271 ± 0.117
$\mathtt{R_{NS}}$	0.551	33.52%	30.991 ± 0.307
R	0.538	32.56%	28.806 ± 0.268

Table 3: Image retrieval (ILSVRC 2012) at TPR=0.9 on S1, $\bar{p}$ is mean % of examples passing S1, time is per dataset.

	mAP	$\bar{p}$	running time [ms]
$\mathtt{D_M}$	0.499	100%	96.711 ± 0.010
$\mathtt{D_{NS}}$	0.479	2.71%	33.752 ± 0.008
D	0.481	2.65%	33.258 ± 0.003

Table 4: Object detection (VOC 2007) at TPR=0.95 on S1, $\bar{p}$ is mean % of examples passing S1, time is per example.

		mAP		$\bar{p}$	running time [sec]
	TPR	full	0.9	0.9	0.9
$\mathtt{R_M}$	C11(64), C5(256), 3×C3(256), C6(4096), C1(4096), C1(4096)	0.587	0.587	100.00%	49.271 ± 0.117
$\mathtt{R^{W3}}$	C11(32/32), C5(128/128), 3×C3(128/128), C6(1024/3072), C1(1024/3072), C1(7/1000)	0.550	0.524	21.38%	29.864 ± 0.191
$\mathtt{R^{W2}}$	C11(16/48), C5(64/192), 3×C3(64/192), C6(512/3584), C1(512/3584), C1(7/1000)	0.558	0.528	26.34%	28.431 ± 0.241
$\mathtt{R^{W1}} = \mathtt{R}$	C11(8/56), C5(32/224), 3×C3(32/224), C6(256/3840), C1(256/3840), C1(7/1000)	0.573	0.538	32.56%	28.806 ± 0.268
$\mathtt{R^{D2}}$	C11(64/-), C5(256/-), 3×C3(256/-), C6(7/4096), C1(-/4096), C1(-/1000)	0.565	0.536	20.76%	45.923 ± 0.189
$\mathtt{R^{D1}}$	C11(64/-), C5(256/-), C3(256/-), C3(7/256), C3(-/256), C6(-/4096), C1(-/4096), C1(-/1000)	0.493	0.470	24.35%	40.448 ± 0.220

Table 5: Image retrieval (ILSVRC 2012) with S1 networks of various width and depth. TPR 'full' allows every example to pass S1. The configuration notation is as in Table 1. Further, if S2 starts deeper or S1 ends shallower, missing layers are indicated by "-" and S1 networks then contain an extra max-pooling layer before their final convolutional layer.

being denoted accordingly. First, we study how the ratio of number of filters allocated to S1 and S2 influences the overall performance. Second, we define a theoretical time complexity and compare it to the empirical running time on a synthetic benchmark.

5.1 Trade-off Analysis

We analyze the effect of reducing the depth or width of $\mathtt{R_M}$ by evaluating networks listed in Table 5. The results convey that this makes S1 weaker, as measured by $\bar{p}$, and S2 stronger, as measured by mAP when S1 is deactivated and all examples pass it (column 'full'). This is expected, as the accuracy of a network is highly dependent on the amount of allocated filters and parameters. Regarding running time, depth reduction brings less benefit than that of width due to lower layers being the most expensive to compute: $\mathtt{R^{D2}}$, mimicking a Faster R-CNN-like cascade, can spare only 6.8% time.

In general, neither sharing nor non-sharing cascades are expected to reach the accuracy of their monolithic baseline due to a non-zero false negative rate at S1. Sharing cascades trade even more accuracy for speed by parameter reduction in S2 (Section 3.1) and shared features serving two different objectives. It can therefore be observed in the column 'full' that none of the OnionNet cascades can achieve the mAP of $\mathtt{R_M}$. To confirm the effect of joint learning, we increased the importance of S2 by retraining R with $\alpha = 0.25$ and obtained improvement of around 0.015 mAP points at an increase of $\bar{p}$ of around 1.5% points.

5.2 Time Cost Analysis

While measuring empirical running time makes for a practical comparison, its generality is limited due to inherent sensitivity to system (esp. GPU architecture) and implementation factors (esp. DNN backend). Thus, we additionally investigate the theoretical time complexity as introduced by He and Sun [15]. The total time complexity of convolutional layers

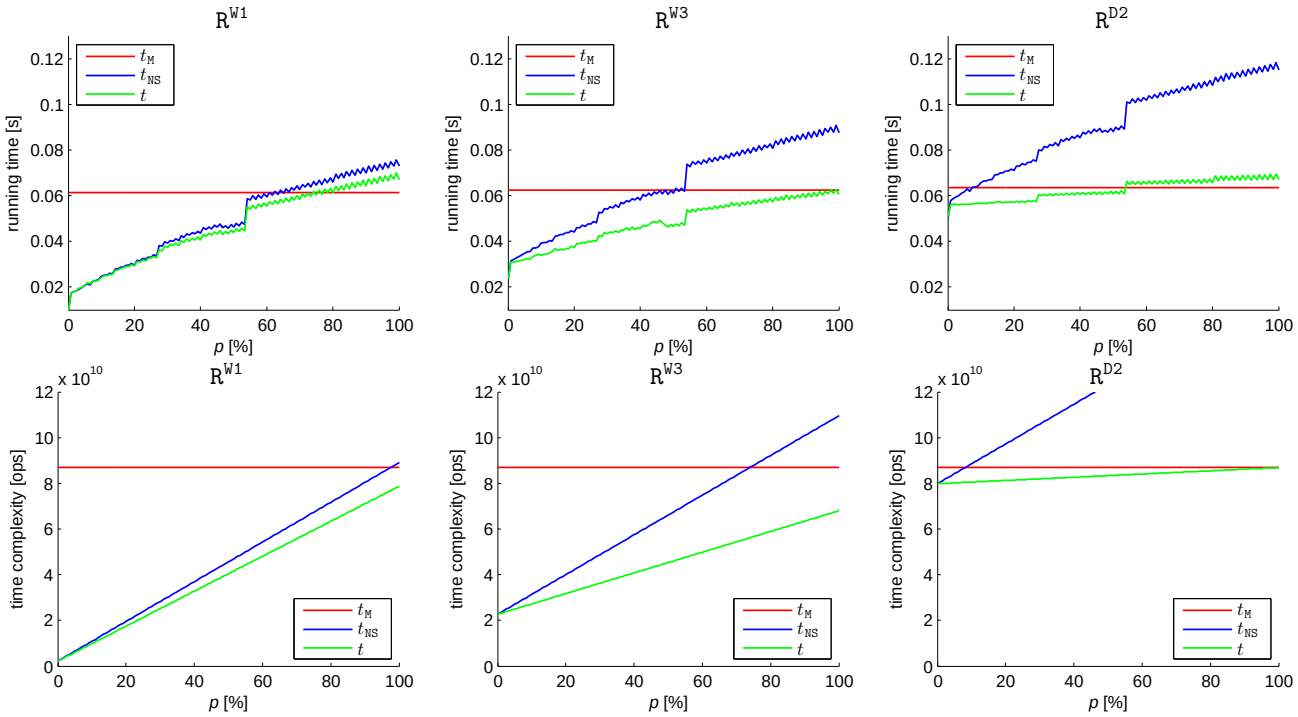

Figure 3: Empirical running times (first row) and time complexities (second row) for $\mathtt{R}_\mathtt{M}$-based cascades as functions of the percentage p of examples passing S1. Sharing cascade (t) is compared with a corresponding non-sharing baseline ($t_\mathtt{NS}$) and its monolithic baseline ($t_\mathtt{M}$).

is defined[2] in the notation of Section 3.1 as $O(\sum_{i=1}^{2}\sum_{l=1}^{d_i} n_{l-1}^{Si} s_l^2 n_l^{Si} m_l^2)$, where d_i is number of convolutional layers in a stage and m_l is the spatial size of an output feature map.

The time cost behavior of a cascade can be best described as a function of the percentage p of examples in a batch passing S1. We plot the costs $t(p)$, $t_\mathtt{M}(p)$, and $t_\mathtt{NS}(p)$ of networks N, $N_\mathtt{M}$, and $N_\mathtt{NS}$ respectively on a synthetic benchmark where we can regulate p as necessary. The disparity $t_\mathtt{NS} - t_\mathtt{M}$ shows for which p a cascade is actually useful and the disparity $t - t_\mathtt{NS}$ reveals the margin of OnionNet due to parameter reduction and feature map sharing. Results for prominent networks of Table 5 are shown in Figure 3 (batch size 120).

Time Complexity. The plots suggests that OnionNet always improves on time cost, as $\forall p : t < t_\mathtt{NS}$. Note that $t_\mathtt{NS} > t_\mathtt{M}$ for higher values of p, *i.e.* non-sharing cascades are overperformed by the monolithic classifiers at some point. The plots also convey that large S1 networks benefit from the speed-up the most ($\mathtt{R}^{\mathtt{W1}}$ vs. $\mathtt{R}^{\mathtt{W3}}$). Also, we can notice that S1 networks of unreduced width are very costly ($\mathtt{R}^{\mathtt{D2}}$) even for small p values, despite the heavy help from feature map sharing.

Running Time. The plots follow the general trend of time complexity plots, although with some important differences. We observe that many configurations perform worse than their monolithic baseline ($t > t_\mathtt{M}$) for large p, except for the configuration with the largest S1 networks ($\mathtt{R}^{\mathtt{W3}}$). Nevertheless, the behavior under smaller p values, *i.e.* those reported in our applications, appears still very promising. We have identified two causes for such an inconsistency between theory and practice, also reported by [11, 18]. One is a nonlinear, nonmonotonous relation of data size and convolution running time due to some sizes being more 'GPU friendly'. The other is the overhead of layer executions, esp. CuDNN kernel launches, since cascades have to basically perform the forward pass twice.

To summarize the analysis, we have shown that our model is theoretically well founded,

[2]This definition does not involve non-convolutional layers and batch reshuffling before evaluating S2: pooling layers "often take 5-10% computational time"[15] and the theoretical complexity of reshuffling is negligible.

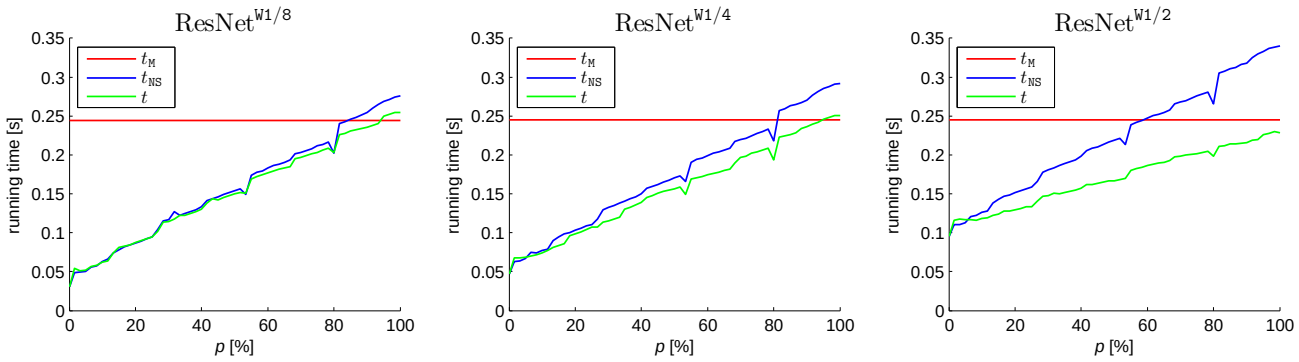

Figure 4: Empirical running times for cascades based on ImageNet ResNet-34 B network [16] created by allocating 1/8, 1/4, or 1/2 of feature channels in each convolutional layer to S1 and leaving the rest in S2; both stages are of the same depth. The notation is identical to Figure 3 and the conclusions from Section 5.2 hold here as well.

although overhead of current GPU solutions has to be considered in practice, which may render small weak classifiers ill-suited for a cascaded solution in general. Due to similar reasons, using more than two stages turned out impractical in our initial experiments. Since the actual benefit of a cascade varies by p, which in practice depends on the precision of S1 at a chosen true positive rate, it was important to identify the sweet spots in practical applications, as we successfully demonstrated above.

6 Conclusion

A novel cascade of feature-sharing deep classifiers was proposed where subsequent stages may be extended by new layers and/or feature channels and their intermediate computations reused. Our motivation was to speed up the evaluation by preventing similar features from being recomputed, which led us to make each stage of the cascade equally deep. Sharing and reduction in model parameters are the main causes of the achieved speed-up. The same factors account for a minor decrease in precision, though. We have demonstrated good speed-ups due to cascades in three important tasks and showed that OnionNet sharing can bring further gain atop of it. We find this fact encouraging, as our applications seem to require some higher-level understanding even for the easy examples, and thus massive speed-ups due to very simple stages as in sliding-window methods should not be expected. As much deeper, more expensive networks are being introduced [16], we believe our method might gain in significance due to larger absolute running time savings; see Figure 4 for a preliminary time cost analysis of a 34-layer residual network cascade.

Acknowledgments. We gratefully acknowledge NVIDIA Corporation for the donated GPU used in this research and Sergey Zagoruyko for providing his early source code for patch matching [37].

References

[1] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron C. Courville. Dynamic capacity networks. *CoRR*, abs/1511.07838, 2015.

[2] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson. Real-time pedestrian detection with deep network cascades. In *BMVC*, 2015.

[3] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *CVPR*, 2005.

[4] Matthew Brown, Gang Hua, and Simon A. J. Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 33 (1):43–57, 2011.

[5] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[6] Ken Chatfield, Karen Simonyan, and Andrew Zisserman. Efficient on-the-fly category retrieval using convnets and GPUs. In *ACCV*, 2014.

[7] Yu Cheng, Felix X. Yu, Rogério Schmidt Feris, Sanjiv Kumar, Alok N. Choudhary, and Shih-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *ICCV*, 2015.

[8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[9] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[11] Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[12] Piotr Dollár, Peter Welinder, and Pietro Perona. Cascaded pose regression. In *CVPR*, 2010.

[13] Ross B. Girshick. Fast R-CNN. In *ICCV*, 2015.

[14] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, 2015.

[15] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[17] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *Deep Learning and Representation Learning, NIPS Workshop*, 2014.

[18] Yani Ioannou, Duncan P. Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving CNN efficiency with hierarchical filter groups. *CoRR*, abs/1605.06489, 2016.

[19] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[20] Hervé Jégou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Pérez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 34(9):1704–1716, 2012.

[21] Zsolt Kira, Raia Hadsell, Garbis Salgian, and Supun Samarasekera. Long-range pedestrian detection using stereo and a cascade of convolutional network classifiers. In *IROS*, 2012.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[23] Chen-Yu Lee, Saining Xie, Patrick W. Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[24] K. Lenc, V. Gulshan, and A. Vedaldi. VLBenchmarks. http://www.vlfeat.org/benchmarks/, 2011.

[25] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E. Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *ICLR*, 2016.

[26] Francisco Massa. Object detection in torch. http://github.com/fmassa/object-detection.torch, 2016.

[27] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(10): 1615–1630, 2005.

[28] Pedro H. O. Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *NIPS*, 2015.

[29] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *NIPS*, 2015.

[30] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014.

[31] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *ICCV*, 2015.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[33] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, 2013.

[34] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[35] Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[36] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. Hd-cnn: Hierarchical deep convolutional neural network for large scale visual recognition. In *ICCV*, 2015.

[37] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *CVPR*, 2015.

[38] Jure Žbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *CoRR*, abs/1510.05970, 2015.

[39] Philipp Zehnder, Esther Koller-Meier, and Luc J. Van Gool. An efficient shared multi-class detection cascade. In *BMVC*, 2008.

[40] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *CoRR*, abs/1505.06798, 2015.

[41] Yefeng Zheng, Liu David, Bogdan Georgescu, Hien Nguyen, and Dorin Comaniciu. 3d deep learning for efficient and robust landmark detection in volumetric data. In *MICCAI*, 2015.