MultiGrain: a unified image embedding for classes and instances

Anonymous authors
Paper under double-blind review

Abstract

We introduce MultiGrain, a neural network architecture that generates compact image embedding vectors that solve multiple tasks of different granularity: class, instance, and copy recognition. MultiGrain is trained jointly for classification by optimizing the cross-entropy loss and for instance/copy recognition by optimizing a self-supervised ranking loss. The self-supervised loss only uses data augmentation and thus does not require additional labels. Remarkably, the unified embeddings are not only much more compact than using several specialized embeddings, but they also have the same or better accuracy. When fed to a linear classifier, MultiGrain using ResNet-50 achieves 79.4% top-1 accuracy on ImageNet, a +1.8% absolute improvement over the the current state-of-the-art AutoAugment method. The same embeddings perform on par with state-of-the-art instance retrieval with images of moderate resolution. An ablation study shows that our approach benefits from the self-supervision, the pooling method and the mini-batches with repeated augmentations of the same image.

1 Introduction

Image recognition is central to computer vision, with dozens of new approaches being proposed every year, each optimized for particular aspects of the problem. From coarse to fine, we may distinguish the recognition of (a) classes, where one looks for a certain type of object regardless of intra-class variations, (b) instances, where one looks for a particular object despite changes in the viewing conditions, and (c) copies, where one looks for a copy of a specific image despite edits. While these problems are in many ways similar, the standard practice is to use specialized, and thus incompatible, image representations for each case.

Consider for example image retrieval, where the goal is to match a query image to a large database of other images, whose applications include detection of copyrighted images and exemplar-based recognition of unseen objects. Often one would like to search the same collection with multiple granularities, by matching the query by class, instance, or copy. Adopting multiple image embeddings, narrowly optimized for each granularity, means multiplying the resource usage. Using a single embedding relevant to all these tasks reduces both the computing time and the storage space. However, this might come at the cost of a reduced accuracy.

In this paper we introduce MultiGrain, a compact embedding that, as illustrated in fig. 1, can solve recognition tasks of different granularities while maintaining or surpassing the accuracy of specialized embeddings. MultiGrain is obtained by training a Convolutional Neural Network (CNN) jointly on the different tasks. CNNs trained for image classification are known to be good universal features extractors. However, authors (Babenko & Lempitsky, 2015) have noted that the intermediate layers of such CNNs are generally better for low-level tasks such as instance and copy recognition. In contrast, our work extracts a single global embedding at the top of the network. The key is to optimize this embedding simultaneously for classification and instance retrieval. In this manner, the same representation integrates different degrees of invariance. Indeed, by definition, copies of the same image contain the same instance, and images that contain the same instance also contain the same class.

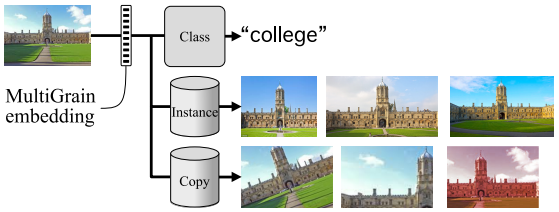
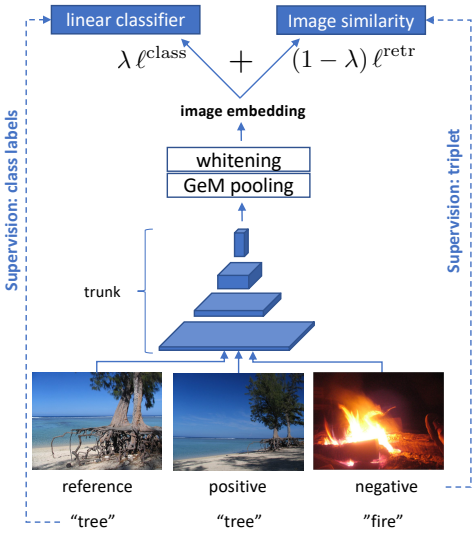


Figure 1: *Top:* Our goal is to extract an image descriptor incorporating different levels of granularity, so that we can solve, classification and particular object recognition tasks: The descriptor is either fed to a linear classifier, or directly compared with cosine similarity.

Right: The MultiGrain architecture.



As an additional contribution, we show that MultiGrain can be learned using only class-level labels via self-supervised learning (Caron et al., 2018). The instance recognition is learned for free, *without labels specific to instance recognition*: we use the identity of arbitrary images as labels, and data augmentation to generate different versions of each image. We also find that, unexpectedly, forming batches with *multiple augmentations of the same image*, improves the classifier performance, even for models trained only for classification. This contradicts the common knowledge that training batches should maximize diversity. Finally, we incorporate in MultiGrain a pooling layer inspired by image retrieval that boosts the classification accuracy for high-resolution images.

Overall, MultiGrain offers compelling performance both for classification and image retrieval, including outperforming the SoTA classification accuracy on ImageNet for ResNet-50.

2 Related work

Image classification. Most CNNs designed for a wide range of tasks leverage a trunk designed for classification, such as Residual networks (He et al., 2016). An improvement on the trunk architecture translates to better accuracies in other tasks (He et al., 2017), see eg. the detection task of LSVRC'15. Architectural improvements (Hu et al., 2018; Huang et al., 2017; Xie et al., 2017) exhibit additional gains; training on weakly annotated data (Mahajan et al., 2018) or using embedding loss at the class level (Wen et al., 2016) can also improve the accuracy. To our knowledge, the state of the art on ILSVRC 2012 for a model learned from scratch on Imagenet data only is currently held by the gigantic AmoebaNet-B architecture (Huang et al., 2018) (557M parameters), which takes 480×480 images as input. In our paper, we choose ResNet-50 (He et al., 2016) (25.6M parameters), as this architecture is adopted in the literature in many works both on image classification and instance retrieval.

Image search. The objective of *Image search* is to find the images most similar to the query in a large image collection. It is usually evaluated for more specific problems such as landmark recognition (Philbin et al., 2007; Jégou et al., 2008), particular object recognition (Nister & Stewenius, 2006) or copy detection (Douze et al., 2009). In this paper *image retrieval* refers to instance-level retrieval, where object instances are as broad as possible, *i.e.*, not restricted to buildings, as in the Oxford/Paris benchmark. Typically, a query image is described by an embedding vector, and the task amounts to searching the nearest neighbors of this vector in the embedding space. Refinement steps include as geometric verification (Philbin et al., 2007), query expansion (Chum et al., 2007; Tolias & Jégou, 2014), or database-side pre-processing or augmentation (Tolias et al., 2016; Turcot & Lowe, 2009), but this paper focuses on the first part. Traditionally, local image descriptors are aggregated to image embeddings, as in the bag-of-words model (Sivic & Zisserman, 2003). It has since become apparent that CNNs trained on classification datasets are competitive image feature extractors for instance retrieval (Babenko et al., 2014; Gong et al., 2014; Razavian et al., 2014).

	classification	retrieval
spatial pool.	avg. pooling	RMAC or GeM
loss	cross-entropy	triplet
batch samp.	diverse	not diverse
whitening	no	yes
resolution	low	high
	(224^2–300^2)	(800–1k×scaled)

Table 1: Differences between classification and image retrieval: Retrieval architectures incorporate a final pooling layer that is regionalized (RMAC of Tolias & Jégou (2014)) or magnifies activations (GeM of Radenović et al. (2018)). The triplet loss (Gordo et al., 2016) requires a batching strategy with pairs of matching images.

Architectures for instance search are regular classification trunks, modified so the pooling stage gives more spatial locality, to cope with small objects and clutter. A competitive baseline for instance retrieval is the R-MAC image descriptor (Tolias et al., 2015). It aggregates regionally pooled features extracted from an activation map. This pooling combined with PCA whitening (Jégou & Chum, 2012) leads to efficient many-to-many comparisons between image regions. Gordo et al. (2016; 2017) fine-tune this representation end-to-end on an external image retrieval dataset. Unlike their approach, we do not assume in this work that we have a domain-specific training set. Radenović et al. (2018) depart from regional pooling by adopting a generalized mean pooling (see section 3.2). It is a spatial pooling of the features raised to an exponent p over the whole image, which offers some benefits as analyzed by Boureau et al. (2010) with respect to noise-to-signal ratio and in simple image classification tasks.

Multi-task training stems from the observation that CNNs transfer to a wide range of vision tasks (Razavian et al., 2014) and exhibit a high level of compressibility (Han et al., 2015). Despite some successes with multi-task networks such as UberNet (Kokkinos, 2017), their design and training still involve numerous heuristics. Ongoing lines of work investigate efficient sharing of parameters (Rebuffi et al., 2018), and proper hyper-parameters settings to weight the gradients from different tasks (Guo et al., 2018).

Data augmentation improves generalization and reduces over-fitting (Krizhevsky et al., 2012). Traditionally, batches were made to contain random samples of the training set. The recently introduced batch augmented (BA) (Hoffer et al., 2019) sampling strategy consists in augmenting the size of the batches and filling them with data-augmented copies of the same image. This yields better generalization performance, and uses computing resources more efficiently through reduced data processing time. We show that this improvement can be obtained *using the same batch size*, *i.e.*, , with a lower number of distinct images per batch. We see this repeated augmentations (RA) scheme as a way to boost the effect of data augmentation over the course of the optimization. Thus, RA is a technique of general interest, beyond large-scale distributed training.

3 Architecture design

In the current best practices, the architectures and training procedures used for class and instance recognition differ significantly. This section describes the differences, summarized in table 1, and our solutions to bridge them, leading to the MultiGrain architecture in fig. 1.

3.1 Training objective

MultiGrain is jointly optimized for the classification and retrieval tasks, which is obtained by combining a classification loss and an instance retrieval loss in the optimization.

Classification loss. We adopt the standard cross-entropy loss. Given $\boldsymbol{e}_i \in \mathbb{R}^d$ the output of eq. (4) for image i, $\boldsymbol{w}_c \in \mathbb{R}^d$ the parameters of a linear classifier[1] for class $c = 1, \ldots, C$, and y_i the ground-truth class for that image, then

$$\ell^{\text{class}}(\boldsymbol{e}_i, [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C], y_i) = -\langle \boldsymbol{w}_{y_i}, \boldsymbol{e}_i \rangle + \log \sum_{c=1}^{C} \exp \langle \boldsymbol{w}_c, \boldsymbol{e}_i \rangle. \tag{1}$$

[1] In practice, we also add a learnable bias term, which is equivalent to adding a constant feature channel to the feature vector $\boldsymbol{e}_i$.

Retrieval loss. The triplet loss (Schroff et al., 2015) imposes that a query image embedding must be closer to the embedding of an image that matches it than to other embeddings. The contrastive loss (Hadsell et al., 2006) imposes a stricter condition: all embedding distances between pairs of matching images must be smaller than all embedding distances between pairs of non-matching images. Optimizing both these losses depends on hard-to-tune hyper-parameters, including how pairs and triplets are sampled.

Some of these issues are solved by the approach of Wu et al. (2017), which starts from a batch of images and (1) normalizes their embeddings to the unit sphere, (2) samples negative pairs using the current embedding similarity, and (3) uses the pairs in a margin loss (that combines contrastive and triplet loss). Given images $i, j \in \mathcal{B}$ in a batch, the margin loss is given by:

$$\ell^{\text{retr}}(\boldsymbol{e}_i, \boldsymbol{e}_j, \beta, y_{ij}) = \max\{0,\ \alpha + y_{ij}(D(\boldsymbol{e}_i, \boldsymbol{e}_j) - \beta)\} \tag{2}$$

where $D(\boldsymbol{e}_i, \boldsymbol{e}_j) = \left\| {}^{\boldsymbol{e}_i}/_{\|\boldsymbol{e}_i\|} - {}^{\boldsymbol{e}_j}/_{\|\boldsymbol{e}_j\|} \right\|$ is the Euclidean distance between the normalized embeddings, the label y_{ij} is equal to $+1$ if the images match and to -1 otherwise, $\alpha > 0$ is the margin (a hyper-parameter), and $\beta > 0$ is a learnable parameter controlling the volume of the space occupied embedding vectors. Due to the normalization, $D(\boldsymbol{e}_i, \boldsymbol{e}_j)$ is equivalent to a cosine similarity, which, up to whitening (section 3.4), is commonly used in retrieval.

We use distance-weighted sampling to sample pairs of images (see appendix A for details). This sampling is suited to our joint training: it tolerates relatively small batch sizes ($|\mathcal{B}| \sim 80$ to 120) and a small amount of positives images (3 to 5) of each instance in the batch, without the need for elaborate parameter tuning or offline sampling.

Joint loss. The joint loss on batch $\mathcal{B}$ is a combination weighted by a factor $\lambda \in [0, 1]$:

$$\frac{\lambda}{|\mathcal{B}|} \cdot \sum_{i \in \mathcal{B}} \ell^{\text{class}}(\boldsymbol{e}_i, \boldsymbol{w}, y_i) + \frac{1-\lambda}{|\mathcal{P}(\mathcal{B})|} \cdot \sum_{(i,j) \in \mathcal{P}(\mathcal{B})} \ell^{\text{retr}}(\boldsymbol{e}_i, \boldsymbol{e}_j, \beta, y_{ij}). \tag{3}$$

Note that the losses are normalized by the number of items in the corresponding summations.

3.2 Spatial pooling operators

For recognition tasks, one requires to encode the whole image as a single vector. The latter is usually obtained by applying a global spatial pooling operator to the 3D activation tensor produced by the convolutional trunk of the network. This should be contrasted with local pooling operators, typically max pooling, that are found throughout the layers of CNNs to achieve local invariance to small translations.

The choice of global pooling operator has a significant effect on the representation. Recent architectures for **classification**, such as ResNet and DenseNet, use average pooling. Average pooling is permutation invariant and hence less sensitive to geometric transformations. It is also flexible as it allows the model to be applied to images of any size.

Image retrieval, on the other hand, requires more localized and fine-grained geometric information than the one captured by average pooling. This is because (i) the representation requires less invariance since object instances and landmarks are visually more similar and (ii) images are often more cluttered, with just a small distinctive part that warrants identification. Hence, the pooling should preserve local information. Next, we discuss the generalized mean pooling operator as a solution to this problem.

Let $\boldsymbol{x} \in \mathbb{R}^{C \times W \times H}$ be the feature tensor computed by a convolutional neural network for a given input image. The tensor represents a feature map with C channels, height H and width W. Let $u \in \Omega = \{1, \dots, H\} \times \{1, \dots, W\}$ be "pixel" in the map, c the channel, and by x_{cu} the tensor element at location u and channel c, so that $\boldsymbol{x} = [x_{cu}]_{c=1,\dots,C, u \in \Omega}$. The *generalized mean pooling* (GeM) layer computes the generalized mean of each channel:

$$\boldsymbol{e} = \left[\left(\frac{1}{|\Omega|} \sum_{u \in \Omega} x_{cu}^p \right)^{\frac{1}{p}} \right]_{c=1,\dots,C} \tag{4}$$

where the exponent $p > 0$ is a parameter. Average pooling and max pooling are equivalent to GeM with $p = 1$, and $p = \infty$, respectively. Exponents in the range $1 < p < \infty$ are a

trade-off between the two (Bo & Sminchisescu, 2009; Boureau et al., 2010; Dollár et al., 2009). GeM was introduced for image retrieval as a component of R-MAC that approximates max pooling (Dollár et al., 2009), but (Radenović et al., 2018) showed it is competitive on its own. (Boureau et al., 2010) studied this layer in the context of scene recognition/image classification. MultiGrain uses it to bridge the two worlds, as well as to dynamically adapt the network to varying image resolution.

3.3 Batching with repeated augmentation (RA)

We introduce *repeated augmentations*, a sampling scheme for training with SGD and data augmentation. In RA we form an image batch $\mathcal{B}$ by sampling $\lceil |\mathcal{B}|/m \rceil$ different images, and transform them up to m times by a set of data augmentations. Thus, the instance level ground-truth $y_{ij} = +1$ iff images i and j are two augmented versions of the same image. The key difference with the standard sampling scheme in SGD is that samples are not independent.

For a given learning rate, RA has lower performance than the standard i.i.d. scheme for small batch sizes, but outperforms it with larger batches. This is different from the observation of (Hoffer et al., 2019), who also consider repeated samples in a batch, but simultaneously increase its size.

With standard sampling, two versions of the same image are seen only in different epochs. We conjecture that correlated RA samples facilitate learning features that are invariant to the only difference between the repeated images — the augmentations. Appendix D shows this phenomenon in a simple artificial setting.

3.4 PCA whitening

We apply a step of PCA whitening to the embeddings to use them for retrieval, in accordance with previous works (Gordo et al., 2017; Jégou & Chum, 2012). The Euclidean distance between transformed features is equivalent to the Mahalanobis distance between the input descriptors. The PCA is trained at the end of the CNN training, using an external dataset of unlabelled images. The whitening operation Φ can be written as $\Phi(\boldsymbol{e}) = \mathbf{S}\left({}^{\boldsymbol{e}}/_{\|\boldsymbol{e}\|} - \boldsymbol{\mu}\right)$ given the whitening matrix $\mathbf{S}$ and centering vector $\boldsymbol{\mu}$.

The parameters of the classification layer have to be modified to take the whitened embeddings as input. For the classifier $\langle \boldsymbol{w}_c, \boldsymbol{e} \rangle$ of eq. (1), we have $\langle \boldsymbol{w}_c, \boldsymbol{e} \rangle = \langle \boldsymbol{w}_c, \Phi^{-1}(\Phi(\boldsymbol{e})) \rangle = \|\boldsymbol{e}\| \left(\langle \boldsymbol{w}'_c, \Phi(\boldsymbol{e}) \rangle + b'_c \right)$ where $\boldsymbol{w}'_c = S^{-\top} \boldsymbol{w}_c$ and $b'_c = \langle \boldsymbol{w}_c, \mu \rangle$ are the modified weight and bias for class c. We observed that inducing decorrelation via a loss (Cogswell et al., 2016) is insufficient to ensure that features generalize well, which concurs with prior works (Gordo et al., 2017; Radenović et al., 2018).

3.5 Input sizes

In image classification, it is standard to resize and center-crop input images to a low resolution, e.g. 224×224 pixels (Krizhevsky et al., 2012). The benefits are a smaller memory footprint, faster inference, and the possibility of batching the inputs if they are cropped to a common size. On the other hand, image retrieval depends on finer details in the images, as an instance can small or seen under a variety of scales. Feature extractors for image retrieval therefore commonly use input sizes of 800 (Gordo et al., 2017) or 1024 (Radenović et al., 2018) pixels, without cropping the image to a square. This is impractical for end-to-end training.

We train MultiGrain at the standard 224×224 resolution, and use larger resolutions at test time. Indeed, a network trained with a pooling exponent p and resolution s can be evaluated at a larger resolution $s^* > s$ using a larger pooling exponent $p^* > p$, see section 4.3.

Proxy task to cross-validate p^*. To select the exponent p^*, suitable for all tasks, we create a synthetic retrieval task **IN-aug**: we sample 2,000 images from the training set of ImageNet, 2 per class, and create 5 augmented copies of each of them. We query all images using the retrieval embeddings and evaluate the retrieval accuracy on IN-aug by measuring

λ	$s^* =$	224	500	800
1	$p^* =$	3	4	4
0.5	$p^* =$	3	4	5

how many of the first 5 augmentations of the image are ranked in top 5 positions. The best-performing $p^* \in \{1, 2, \ldots, 10\}$ on IN-aug is shown in the table.

The optimal p^* obtained on IN-aug is a trade-off between retrieval and classification. Experimentally, we observed that other choices are suitable for setting this parameter: fine-tuning the p^* using training inputs at a given resolution and back-propagating the cross-entropy loss provides similar results and values of p^* (but is more complex).

4 Experiments and Results

4.1 Experimental settings

Base architecture. We build MultiGrain using ResNet-50 as convolutional trunk (He et al., 2016). The latter is optimized using SGD, starting with a learning rate of 0.2 which is reduced tenfold at epochs $30, 60, 90$ for a total of 120 epochs (a standard setting (Paszke et al., 2017)). The batch size is $|\mathcal{B}| = 512$ and an epoch "sees" a fixed number $T = 5005$ batches. With uniform sampling, one epoch does two passes over the training set; with RA and $m = 3$, one epoch sees $\sim 2/3$ of the images of the training set. The baselines are trained using this longer schedule for a fair comparison.

Data augmentation. Use a standard set of data augmentations (Howard, 2013) detailed in the appendix (table E.1); we refer to this set of augmentations as "full". The baseline CNN reaches 76.2% top-1 validation error when trained with cross-entropy alone and uniform batch sampling (see table 2). This is on the high end of accuracies reported for the ResNet-50 network (Goyal et al., 2017; He et al., 2016) without specially-crafted regularization terms (Zhang et al., 2018), data augmentations (Cubuk et al., 2018) or external data.

Pooling exponent. During the training of our network, we consider two settings for the GeM layer of section 3.2: we set either $p = 1$ or $p = 3$. Related work (Radenović et al., 2018) and our preliminary experiments suggest that the value $p = 3$ improves the retrieval performance. Appendix B illustrates the effect of this choice.

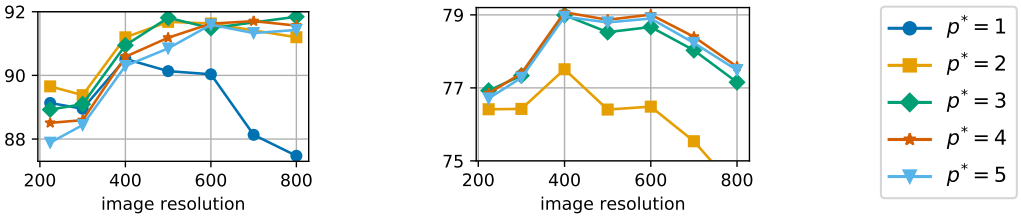
Input size and cropping. As described in section 3.5, we train our network on crops of 224×224 pixels. For testing, we experiment with resolutions $s^* = 224, 500, 800$. For resolution $s^* = 224$, we follow the usual classification protocol: the smallest side of the image is resized to 256 and then a 224×224 central crop is extracted. For resolution $s^* > 224$, we instead follow the protocol common in image retrieval: resize the largest side of the image to s^* and evaluate the network on the rectangular image without cropping.

Margin loss and batch sampling. We use $m = 3$ RA repetitions per batch. We use the default margin loss hyperparameters of (Wu et al., 2017) (see appendix E). As in (Wu et al., 2017) distance-weighted sampling is performed independently on each of the 4 GPUs used for training.

Datasets. We train our networks on the ImageNet-2012 training set. Classification accuracies are reported on the validation images. For image retrieval, we report the mean average precision on the **Holidays** dataset (Jégou et al., 2008), with images rotated manually when necessary, as in prior evaluations (Gordo et al., 2016). We also report the accuracy on the **UKB** object recognition benchmark (Nister & Stewenius, 2006), which shows 2,550 objects under 4 viewpoints each; each image is used as a query to find its 4 closest neighbors in embedding space; the number of correct neighbors is averaged across all images (*i.e.*, the score is in $[0, 4]$). We report the performance of our network in a copy detection setting, indicating the mean average precision on the "strong" subset of the Inria Copydays dataset (Douze et al., 2009), combined with 10k distractor images randomly sampled from YFCC100M (Thomee et al., 2016). We call the combination **C10k**. The PCA whitening transformations are computed from the features of 20k images from YFCC100M, distinct from the C10k distractors.

4.2 Effect of the pooling exponent p^* and the loss weighting λ

As a starting point, we use RA sampling and pooling exponent $p = 3$. This gives a 76.9% top-1 validation accuracy on ImageNet, 0.7% points above the baseline, see table 2.



(a) Holidays (mAP) (b) ImageNet val (top-1)

Figure 2: Retrieval and classification accuracies as a function of pooling exponent p^* and the image resolution. At training time, the pooling was $p = 3$. Note the clear interaction between the resolution s^* and the pooling exponent p^*.

We now use larger images at test time, *i.e.*, , we set $s^* > 224$ and vary the exponents $p^* \neq p = 3$. Figures 2a and 2b show the classification accuracy and the retrieval accuracy at different resolutions, for different values of the exponent p^*. As expected, at $s^* = 224$, the pooling exponent yielding best accuracy in classification is the exponent with which the network has been trained, $p^* = 3$; instead, testing at larger scale requires an exponent $p^* > p$, both for classification and for retrieval. In the following, we adopt the values obtained by our cross-validation on IN-aug, see section 3.5.

We defer to appendix C for the study on the weighting parameter λ. We set $\lambda = 0.5$ in our following experiments, as it gives the best classification accuracy at the practical resolutions $s^* = 224$ and 500 pixels. As a reference, we also report a few results with $\lambda = 1$ (*i.e.*, ignoring the retrieval loss).

4.3 Classification results

From now on, our MultiGrain nets are trained at resolution $s = 224$ with exponent $p = 1$ or $p = 3$ in the GeM pooling. For each evaluation resolutions $s^* = 224, 500, 800$, the same exponent p^* is selected according to section 3.5, yielding a single embedding for classification and for retrieval. Table 2 presents the classification results. There is a large improvement in classification performance from our baseline ResNet-50 with $p = 1$, $s = 224$, "full" data augmentation (76.2% top-1 accuracy), to a MultiGrain model at $p = 3$, $\lambda = 0.5$, $s = 500$ (78.6% top-1). We identify four sources for this improvement:

1. The RA batch sampling (section 3.3) yields an improvement of +0.6% ($p = 1$).
2. The retrieval loss helps the generalizing effect of data augmentation: +0.2% ($p = 1$).
3. $p = 3$ pooling: GeM at training (section 3.2) allows the margin loss to have a much stronger effect thanks to increased localization of the features: +0.4%.
4. Expanding resolution: evaluating at resolution 500 adds +1.2% to the $p = 3$ MultiGrain network, reaching the 78.6 top-1 accuracy. The $p = 3$ training yields sparser features, more generalizable over different resolutions, and the p^* pooling adaptation (without it the performance at this resolution is only 78.0%).

The p^* selection for evaluation at higher resolutions has its limits: at 800 pixels, due to the large discrepancy between the training and testing scale for the feature extractor, the accuracy drops to 77.2% (76.2% without the p^* adaptation).

AutoAugment (AA) is a reinforcement learning approach to find data augmentations that improve the accuracy of CNNs (Cubuk et al., 2018). We integrate the augmentations found on their ResNet-50 model. To give more impact to AA, we do 270 passes over the dataset, with batch size 512. MultiGrain with AA reaches 78.2% top-1 accuracy at $s^* = 224$ ($p = 3$, $\lambda = 0.5$). To the best of our knowledge, this is the state-of-the-art for ResNet-50 when evaluating at this resolution: it outperforms AA alone (77.6%) and mixup (Zhang et al., 2018) (76.7%). Increasing the test resolution improves the accuracy to 79.4% at $s^* = 500$.

We also experimented with other architecture, see appendix G. We observed that the GeM pooling substantially increase the accuracy *of off-the-shelf networks*, with only a tiny fine-

Architecture	λ	data aug.	resol. s^*	train-time pooling $p=1$	train-time pooling $p=3$
ResNet-50		full	224	76.2 / 92.9	76.2 / 93.1
MultiGrain	1	full	224	76.8 / 93.2	76.9 / 93.5
MultiGrain	0.5	full	224	77.0 / 93.6	**77.4** / **93.6**
MultiGrain	0.5	AA	224	77.4 / 93.6	**78.2** / **93.9**
MultiGrain	0.5	full	500	76.5 / 93.5	**78.6** / **94.4**
MultiGrain	0.5	AA	500	77.7 / 94.0	**79.4** / **94.8**
MultiGrain	0.5	full	800	73.5 / 93.5	77.2 / 93.5
MultiGrain	0.5	AA	800	74.1 / 91.8	77.8 / 93.9
PyTorch model zoo			224	76.1 / 92.9	
mixup			224	76.7 / 94.4	
BA ($\|\mathcal{B}\| = 1024$)			224	76.9 / –	
AutoAugment			224	77.6 / 93.8	

Table 2: ImageNet 2012 validation performance at top-1 / top-5 accuracies (%). Resnet-50 is a classification baseline trained with cross-entropy with our training schedule, data augmentation, and uniform batch sampling. MultiGrain uses the same Resnet-50 trunk. At resolutions $s^* > 224$ we evaluate with exponent p^* as described in section 3.5. We compare mixup (Zhang et al., 2018), BA (Hoffer et al., 2019), and AutoAugment (Cubuk et al., 2018).

Method	resol. s^*	Holidays	UKB	CD10k
MultiGrain $\lambda = 1$	500	**91.8**	3.89	**81.1**
MultiGrain $\lambda = 1$	800	91.6	**3.91**	**82.5**
MultiGrain $\lambda = 0.5$	500	91.5	**3.90**	80.7
MultiGrain $\lambda = 0.5$	800	**92.5**	**3.91**	78.6
Fisher vectors	800	63.4	3.35	42.7
Neural codes	224	79.3	3.56	
ResNet-50 RMAC	724	90.9		
ResNet-50 RMAC	1024	93.3		
ResNet-101 RMAC	800	91.4	3.89	
GeM†	**1024**	**93.9**		

Table 3: Instance search results and baselines, on Holidays (% mAP) and UKB (/4). We set $p = 3$ pooling at training time for our MultiGrain models, and p^* set as given in section 3.5. We compare Fisher vectors (Jégou et al., 2012), neural codes (Babenko et al., 2014), RMAC (Gordo et al., 2016), and GeM (Radenović et al., 2018). † GeM is fine-tuned at resolution 362×362 on additional retrieval data and uses multi-scale input processing at an extra cost.

tuning. For example, we obtain a top-1 accuracy of 83.6 with a PNASNet-5-Large, a +0.9% improvement over the original (Liu et al., 2018).

4.4 Retrieval results

Retrieval results are in table 3, an ablation study and copy detection results are in the F. Our MultiGrain nets improve accuracies on all datasets with respect to the ResNet-50 baseline for comparable resolutions. Repeated augmentations (RA) is again a key ingredient in this context. We compare with reported accuracies in (Gordo et al., 2016; 2017), without additional training data. MultiGrain compares favorably with their results at the same resolution ($s^* = 800$). They reach accuracies above 93% mAP on Holidays but this requires a resolution $s \geq 1000$ pixels.

Note that MultiGrain reaches a reasonable retrieval performance at resolution $s^* = 500$, an interesting operating point compared to the traditional inference resolutions $s = 800$–1000 for retrieval. Indeed, a forward pass of ResNet-50 on 16 processor cores takes 3.80s at resolution 500, against 18.9s at resolution 1024 (5× slower). Because of this quadratic increase in timing, and the single embedding computed by MultiGrain, our solution is particularly apt in large-scale or low-resource vision applications. At resolutions 500 the results with margin loss ($\lambda = 0.5$) are slightly lower than without ($\lambda = 1$). This is partly due to the limited transfer from the IN-aug task to the variations observed in retrieval datasets.

5 Conclusion

MultiGrain is a unified embedding for image classification and instance retrieval. It relies on a classical CNN trunk, with a GeM pooling layer, topped with two heads at training time. We have discovered that this pooling layer allows us to increase the resolution of images used at inference time, while maintaining a small resolution at training time. We have shown that MultiGrain embeddings can perform well on classification and retrieval. Interestingly, MultiGrain also sets a new state of the art on pure classification compared to all results obtained with the same convolutional trunk. Our approach will be open-sourced.

References

A. Babenko and V. Lempitsky. Aggregating deep convolutional features for image retrieval. In *Proc. ICCV*, 2015. 1

Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Proc. ECCV*, pp. 584–599. Springer, 2014. 2, 8, 16

Liefeng Bo and Cristian Sminchisescu. Efficient match kernel between sets of features for visual recognition. In *Proc. NIPS*, 2009. 5

Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proc. ICML*, 2010. 3, 5, 12

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proc. ECCV*, 2018. 2

Ondrej Chum, James Philbin, Josef Sivic, Michael Isard, and Andrew Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *Proc. ICCV*, 2007. 2

Michael Cogswell, Faruk Ahmed, Ross B. Girshick, C. Lawrence Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *CoRR*, abs/1511.06068, 2016. 5

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv:1805.09501*, 2018. 6, 7, 8

Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. In *Proc. BMVC*, 2009. 5

Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of gist descriptors for web-scale image search. In *Proc. CIVR*, 2009. 2, 6

Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *Proc. ECCV*, pp. 392–407. Springer, 2014. 2

Albert Gordo, Jon Almazán, Jérôme Revaud, and Diane Larlus. Deep image retrieval: Learning global representations for image search. In *Proc. ECCV*, 2016. 3, 6, 8

Albert Gordo, Jon Almazán, Jérôme Revaud, and Diane Larlus. End-to-end learning of deep visual representations for image retrieval. *IJCV*, 124:237–254, 2017. 3, 5, 8

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv:1706.02677*, 2017. 6

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *Proc. ECCV*, pp. 282–299. Springer, 2018. 3

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proc. CVPR*, 2006. 4

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015. 3

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016. 2, 6

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proc. ICCV*, 2017. 2

Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: better training with larger batches. *arXiv e-prints*, art. arXiv:1901.09335, January 2019. 3, 5, 8

Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv:1312.5402*, 2013. 6

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proc. CVPR*, 2018. 2, 16

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. CVPR*, 2017. 2

Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv:1811.06965*, 2018. 2

Hervé Jégou and Ondřej Chum. Negative evidences and co-occurences in image retrieval: The benefit of pca and whitening. In *Proc. ECCV*, pp. 774–787. Springer, 2012. 3, 5

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proc. ECCV*, 2008. 2, 6

Hervé Jégou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *PAMI*, 34(9), 2012. 8

Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *Proc. CVPR*, pp. 5454–5463, 2017. 3

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pp. 1097–1105, 2012. 3, 5

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proc. ECCV*, pp. 19–34, 2018. 8, 16

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proc. ECCV*, 2018. 2

David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, 2006. 2, 6

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Autodiff Workshop*, 2017. 6

J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proc. CVPR*, 2007. 2

Filip Radenović, Giorgos Tolias, and Ondrej Chum. Fine-tuning CNN image retrieval with no human annotation. *TPAMI*, 2018. 3, 5, 6, 8

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. *Proc. CVPR Workshop*, pp. 512–519, 2014. 2, 3

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proc. CVPR*, pp. 8119–8127, 2018. 3

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proc. CVPR*, pp. 815–823, 2015. 4

Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. ICCV*, 2003. 2

Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: the new data in multimedia research. *Commun. ACM*, 59:64–73, 2016. 6

Giorgos Tolias and Hervé Jégou. Visual query expansion with or without geometry: Refining local descriptors by feature aggregation. *Pattern Recognition*, 47(10), 2014. 2, 3

Giorgos Tolias, Ronan Sicre, and Hervé Jégou. Particular object retrieval with integral max-pooling of cnn activations. *CoRR*, abs/1511.05879, 2015. 3

Giorgos Tolias, Yannis Avrithis, and Hervé Jégou. Image search with selective match kernels: aggregation across single and multiple images. *IJCV*, 116(3):247–261, 2016. 2

Panu Turcot and David G Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. In *Proc. ICCV*, 2009. 2

Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *Proc. ECCV*, 2016. 2

Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Sampling matters in deep embedding learning. In *Proc. ICCV*, 2017. 4, 6, 12

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *Proc. CVPR*, pp. 5987–5995, 2017. 2

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *Proc. ICLR*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb. 6, 7, 8

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proc. CVPR*, pp. 8697–8710, 2018. 16

Appendix

We report a few details and additional experiments that did not fit in the main paper. Appendix A outlines the repeated augmentation sampling algorithm. Appendix B illustrates the effect of GeM pooling on activation maps. Appendix C studies the effect of the loss weighting parameter. Appendix D shows the effect of data-augmented batches when training a simple toy model. Appendix E lists the values of a few hyper-parameters used in our method. Appendix F gives a some more ablation results in the retrieval setting. Finally, Appendix G shows how to use the ingredients of MultiGrain to improve the accuracy of an off-the-shelf pre-trained ConvNet at almost no additional training cost. It obtains what appear to be **the best reported classification results on imagenet-2012** for a convnet with publicly available weights.

A Sampling pairs in image batches

We formalize the algorithm used to sample batches with repeated augmentations.

The loss of eq. (2) is computed on a subset of positive and negative pairs $\mathcal{P}(\mathcal{B}) \subset \mathcal{B}^2$ obtained as $\mathcal{P}(\mathcal{B}) = \mathcal{P}_+(\mathcal{B}) \cup \mathcal{P}_-(\mathcal{B})$ where (Wu et al., 2017):

$$\mathcal{P}_+(\mathcal{B}) = \{(i,j) \in \mathcal{B}^2 : y_{ij} = 1\}, \qquad \mathcal{P}_-(\mathcal{B}) = \bigcup_{(i,j)\in\mathcal{P}_+} \{(i, j^*) \text{ with } j^* \sim p(\cdot|i)\}. \tag{A.1}$$

This means that one retains all positive pairs in the batch and then, for each positive pair (i,j), generates a negative pair (i,j^*) by sampling j^* with probability

$$p(j|i) \propto \min\{\tau, q^{-1}(D(\boldsymbol{e}_i, \boldsymbol{e}_j))\} \cdot \mathbf{1}_{\{y_{ij}=-1\}},$$

where $\tau > 0$ is a parameter and $q(z) \propto z^{d-2}(1-z^2/4)^{\frac{d-3}{2}}$ is a PDF that depends on the embedding dimension d.

B Illustration of the effect of p^*

We visualize the effect of changing the GeM pooling exponent p on activation maps at different resolutions. We focus on a single class (racing car) and make the simplistic assumption that there is one channel of the activation map that reacts strongly to that class.

Then we can visualize the activation map for that channel on images. Figure B.1 shows a typical result. By setting $p = 3$, the car is detected with high confidence and without spurious detections. Boureau et al. (2010) analyse average- and max-pooling of sparse features. They find that when the number of pooled features increases, it is beneficial to make them more sparse, which is consistent with the observation we make here.

C Analysis of the tradeoff parameter

We analyze the impact of the tradeoff parameter λ between the two components of the loss of eq. (3). Note, this parameter does not directly reflect the relative importance of the two loss terms during training, since these are not homogeneous: $\lambda = 0.5$ does not mean that they have equal importance.

Figure C.1 analyzes the actual relative importance of the classification and margin loss terms, by measuring the average norm of the gradient back-propagated through the network at epochs 0 and 120. One can see that $\lambda = 0.5$ means that the classification has slightly more weight at the beginning of the training. The classification term becomes dominant at the end of the training, meaning that the network has already learned to cancel data augmentation.

In terms of performance, $\lambda = 0.1$ leads to a poor classification accuracy. Interestingly, the classification performance is higher for the intermediate $\lambda = 0.5$ (77.4% at $s^* = 224$) than for $\lambda = 1$, see Table 2. Thus, the margin loss leads to a performance gain for the classification task.

We set $\lambda = 0.5$ in our following experiments, as it gives the best classification accuracy at the practical resolutions $s^* = 224$ and 500 pixels. As a reference, we also report a few results with $\lambda = 1$.

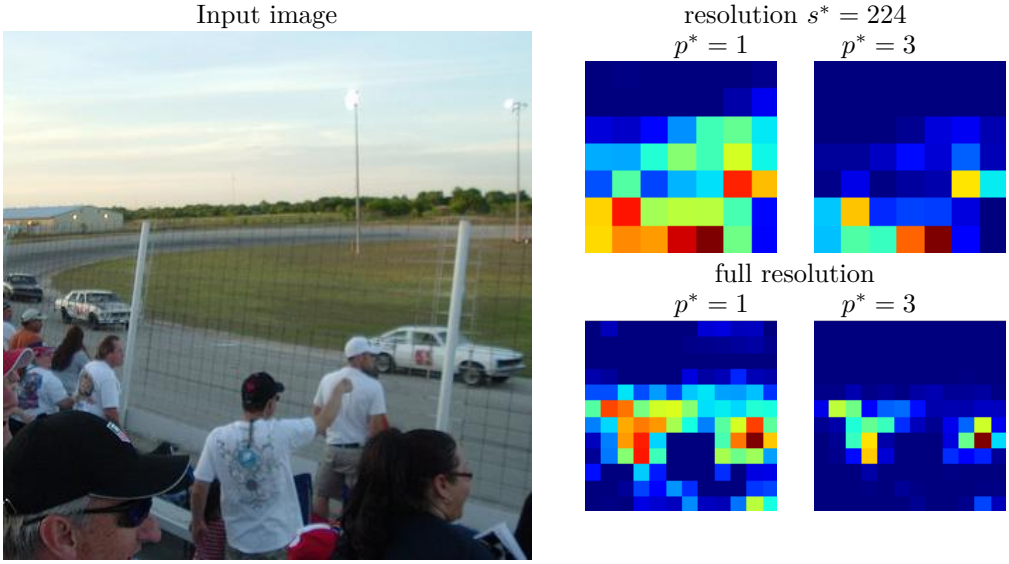


Figure B.1: An off-the-shelf ResNet-50 reacts strongly on channel 909 of the last activation map for class "racing car". The image on the left is a hard example for the class. We show channel 909 for that image, at several resolutions and with GeM parameters $p^* = 1$ and $p^* = 3$. In the low resolution version, the cars are too small to be visible individually on the activation map. In the full resolution version, the location of the cars is more clear. In addition, $p^* = 3$ reduces the noisy detections relative to the true locations.

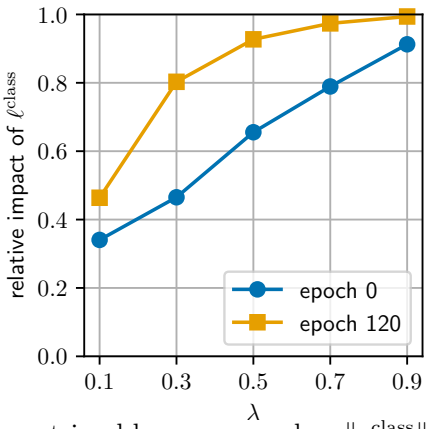


Figure C.1: Classification vs retrieval loss, measured as $\|g^{\text{class}}\|/(\|g^{\text{class}}\| + \|g^{\text{retr}}\|)$, where the g^{class} vector is the gradient from the $\lambda\ell^{\text{class}}$ component.

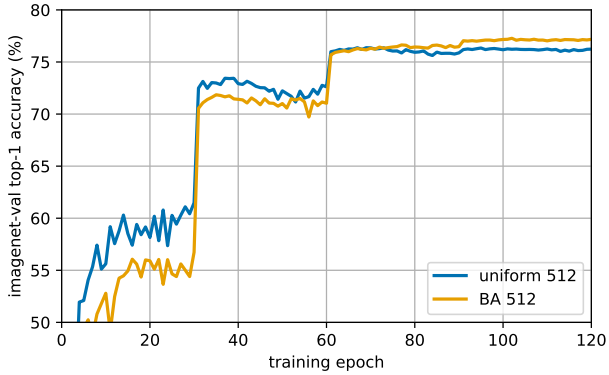


Figure D.1: Evolution of the validation accuracy on ImageNet-val with and without data-augmented batches.

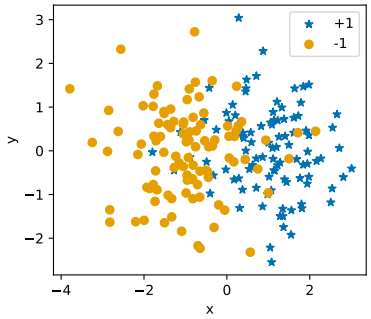


Figure D.2: Training set for the toy model in appendix D.

D Data-augmented batches: toy model

We have observed in section 3.3 and appendix C that training our architecture (ResNet-50 trunk) with data-augmented batches yields improvements with respect to the vanilla uniform sampling scheme, despite the decrease in image diversity.

This observation holds even in the absence of ranking triplet loss, all things being equal otherwise: same number of iterations per epoch, number of epochs, learning rate schedule, and batch size. As an example, fig. D.1 shows the evolution of the validation accuracy of our network trained under cross-entropy with our training schedule and a $p = 1$ pooling, batches of size 512, with the data augmentation introduced in section 4.1, with uniform batches vs. with batch sampling. While initial epochs suffer from the reduced diversity of the batches compared to the uniformly-sampled variant, the reinforced effect on data augmentation compensates for this in the long run, and makes the batch-augmented variant reach a higher final accuracy.

Since we observe this better performance even for a pure image classification task, an interesting question is whether this benefit is specific to our architecture and training method (batch-norm, etc), or if it is more generally applicable? Hereafter we analyse a linear model and synthetic classification task that seems to align with the second hypothesis.

We consider an idealized model of the effect of including different data-augmented instances of the same image in one batch using standard stochastic gradient descent. We create a synthetic training set $\mathcal{D}$ of points pictured in fig. D.2 of $N = 100$ positive and $N = 100$ negative training points $\boldsymbol{p^i} = (p^i_x, p^i_y)$ by sampling from two 2D Gaussian distributions:

$$\begin{aligned} p^i_x &\sim \mathcal{N}(\mu = 0, \sigma = 1) \\ p^i_y &\sim \mathcal{N}(\mu = y^*_i, \sigma = 1) \end{aligned} \tag{D.1}$$

with $y^*_i = \pm 1$ being the ground truth label. We sample a test dataset in the same manner.

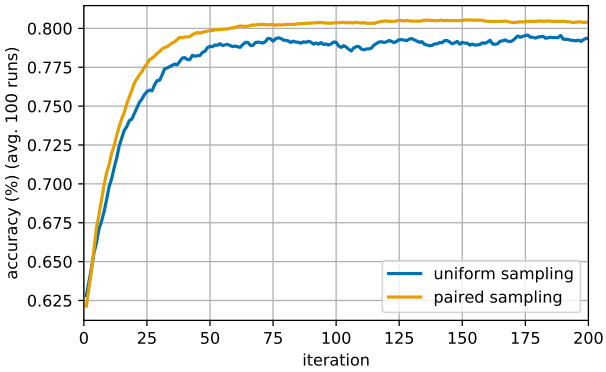


Figure D.3: Evolution of the test accuracy of the SVM trained on the synthetic data, averaged accross 100 runs.

Table E.1: Margin loss and data-augmentation parameters

parameter	value
margin α	0.2
initial β_0	1.2
β learning rate	0.1

We consider the SGD training of an SVM

$$f_{\boldsymbol{w}}(\boldsymbol{p_i}) = \boldsymbol{w}^\top \boldsymbol{p_i} \tag{D.2}$$

using the Hinge loss

$$\ell^{\text{hinge}} = \max\left(1 - y_i^* f_{\boldsymbol{w}}(\boldsymbol{p_i}), 0\right). \tag{D.3}$$

We consider the symmetry across the x-axis

$$\phi((p_x^i, p_y^i)) = \phi((p_x^i, -p_y^i)) \tag{D.4}$$

as a label-preserving data-augmentation suited to our synthetic dataset. We train the SVM (equation D.2) using one pass through the data-augmented dataset $\bar{\mathcal{D}}$ of size $4N$, using batches of size 2.

The only difference between the two optimization schedules is the order in which the samples are batched and presented to the optimizer. We consider two batch sampling strategies:

- Uniform sampling: we sample the elements of the batch randomly from $\bar{\mathcal{D}}$, without replacement;
- Paired sampling: we generate a batch by pairing a random element from $\bar{\mathcal{D}}$ and its data-augmentation, removing these two elements from $\bar{\mathcal{D}}$.

Figure D.3 shows the evaluation of the accuracy with the iterations in both of these cases, averaged across 100 runs. It is clear that pairing the data-augmented pairs in one batch accelerates the convergence of this model.

This idealized experiment demonstrates that there are cases in which the repeated augmentation scheme provides an optimization and generalization boost, and reinforces the effect of data augmentation.

E Margin loss hyper-parameters

Table E.1 gives the value of the hyper-parameters for the margin loss used during the training of our models.

Table E.2 gives the transformations in the *full* data augmentation used in our experiments (section 4.1), along with their parameters.

Table E.2: *full* data-augmentation transforms and parameters

transformation	**parameter range**
horizontal flip	
random resized crop	scale $\in [0.08, 1.0]$ ratio $\in [3/4, 4/3]$
color jitter	brightness 0.3 contrast 0.3 saturation 0.3
lighting transform	intensity 0.1

Table D.1: Full results including Copydays + 10k distractors (CD10k, % mAP), and ablation study for the MultiGrain models. The Pytorch model simply extract the last activation layer as a descriptor (Babenko et al., 2014). Resnet-50 corresponds to features extracted from a classification baseline with $p = 1$ or $p = 3$ GeM pooling, trained with cross-entropy with our training schedule, data augmentation, and uniform batch sampling.

			Holidays			UKB			CD10k		
Method	λ	$s^* =$	224	500	800	224	500	800	224	500	800
PyTorch model zoo			85.5	86.6	82.8	3.71	3.85	3.80	61.5	61.1	43.0
Resnet-50 trained with $p = 1$ pooling			83.5	88.8	87.1	3.60	3.79	3.82	59.2	69.9	66.2
Resnet-50 trained with $p = 3$ pooling			86.8	90.0	90.4	3.73	3.87	3.89	70.6	78.9	75.7
MultiGrain	1		**88.9**	**91.8**	91.6	**3.78**	3.89	**3.91**	**75.1**	**81.2**	**82.5**
MultiGrain	0.5		88.3	91.5	**92.5**	**3.78**	**3.90**	**3.91**	74.1	80.7	78.6
MultiGrain + AA	0.5		86.5	90.3	89.4	3.75	3.89	3.90	69.7	77.8	76.1

F Additional results and ablation study for Multigrain in retrieval

Table D.1 reports additional results of the MultiGrain architecture, with an ablation study analyzing the effect of each component.

As already reported in the main paper, for some datasets the choice of not using the triplet loss ($\lambda = 1$) is as good or better than our generic choice ($\lambda = 0.5$). Of course, then the embedding is not multi-purpose anymore. Overall, the different elements employed in our architecture (RA and the layers specific to Multigrain) still give a significant improvement over simply using the activations, and is competitive with the state of the art for the same resolution/complexity.

Note, the AutoAugment data augmentation does not transfer well to the retrieval tasks. This can be explained by their specificity to Imagenet classification. This shows the limitation of a particular choice of data-augmentation if a single embedding for classification and retrieval datasets is desired. Learning AutoAugment specifically for the retrieval task would certainly help, but would probably also result in less general embeddings. Hence, data-augmentation is a limiting factor for multi-purpose embeddings: improving for one task like classification hurts the performance for other tasks.

Table E.1: Additional top-1/top-5 validation classification accuracies obtained by finetuning p^* for higher evaluation scales from off-the-shelf networks: NASNet (Zoph et al., 2018), SENet (Hu et al., 2018) and PNASNet (Liu et al., 2018). The first column indicates the training resolution s and the accuracy we measured at this resolution, with standard evaluation (resize of the largest scale to $s \cdot 256/224$ + center crop). The subsequent columns show the accuracy measured at higher resolutions $s^* = 350, 400, 450, 500$ without cropping, together with the p^* found by finetuning for these resolutions (appendix G).

	original evaluation		$s^* = 350$		$s^* = 400$		$s^* = 450$		$s^* = 500$	
Architecture	s	acc. (%)	p^*	acc. (%)	p^*	acc. (%)	p^*	acc. (%)	p^*	acc. (%)
NASNet-A-Mobile	224	74.1/91.7	1.7	**75.1**/**92.5**	2.1	74.2/92.1	2.4	71.8/90.9	2.6	68.4/89.0
SENet154	224	81.3/95.5	1.6	82.6/96.2	1.6	83.0/96.5	1.6	**83.1**/**96.5**	1.7	82.7/96.3
PNASNet-5-Large	331	82.7/96.0	1.0	81.3/85.4	1.4	82.6/96.1	1.5	83.2/96.4	1.7	**83.6**/**96.7**

G Evaluation of off-the-shelf classifiers at higher resolutions

In this section, we present some additional classification results using off-the-shelf pretrained classification networks trained with standard average pooling ($p = 1$).

As outlined in sections 3.5 and 4.2, one of our contributions is a strategy for evaluating classifier networks trained with GeM pooling at scale s and exponent p at a higher resolution s^* and adapted exponent p^*. It can be used on pretrained networks as well.

For an evaluation scale s^*, we use the alternative strategy described in section 3.5 to choose p^*: we finetune the parameter p^* by stochastic gradient descent, backpropagating the cross-entropy loss on training images from imagenet, rescaled to the desired input resolution. Compared to a full finetuning at this input resolution, this strategy has a limited memory footprint, given that the backpropagation only has to be done on the ultimate classification layer before reaching the pooling layer, allowing for an efficient computation of the gradient of p^*. Experimentally we also found that this process converges on a few thousands of training samples, while a finetuning of the classification layer would require several data-augmented epochs on the full training set.

The finetuning is done using SGD with batches of $|\mathcal{B}| = 4$ (non-cropped) images, with momentum 0.9 and initial learning rate $\mathrm{lr}^{(0)} = 0.005$, decayed under a polynomial learning rate decay

$$\mathrm{lr}^{(i)} = \mathrm{lr}^{(0)} \left(1 - \frac{i}{i_{\max}}\right)^{0.9} \tag{G.1}$$

with $i_{\max}$ the total number of iterations.

We select $50,000$ images from the training set (50 per category) for the fine-tuning and do one pass on this reduced dataset. We use off-the-shelf pretrained convnets from the Cadene/pretrained-model repository[2]. Table E.1 outlines the resulting validation accuracies. We see that for each network there is a scale and choice of p^* that performs better than the standard evaluation.

These networks have not been trained using GeM pooling with $p > 1$; as exhibited in our classification results (table 2) we found this to be another key ingredient in ensuring a higher scale insensitivity and better performance at larger resolution. As in our main experiments with the MultiGrain architecture with a ResNet-50 backbone, it is likely that these networks would reach higher values when training from scratch with a $p > 1$ pooling, and adding repeated augmentations and margin loss. However, running training experiments on these large networks is significantly more expensive. Therefore, we leave this for future work.

[2]Url: https://github.com/Cadene/pretrained-models.pytorch