Status report - Deep learning with Siamese networks for instance search or identification

Matthias Kohl

March 20, 2017

This report serves as a status report for the internship on Siamese networks for instance search (image retrieval), supervised by Georges Quénot and Jean-Pierre Chevallet, in collaboration with Maxime Portaz.

This report should not be seen as a scientific report. Instead, it is a simple overview of the main ideas and results of the internship so far and will be updated on the fly.

1 Introduction

We consider the problem of instance search for images. The goal is as follows: given a reference dataset of images and a query image, retrieve the image from the reference dataset that best suits the query image.

This problem is related to image classification: if we consider each instance of an object as a class, we can classify the query image and return any image from that class in the reference dataset.

The difference with image classification is that there is almost no variability within the classes, as they are in fact a single object. So the objective is to focus more on differentiating between objects, rather than predicting the type of object. Thus for a given query image, we try to tell how much it resembles each image of the reference dataset.

2 State of the art

Previously, most systems were based on a bag-of-words approach in order to match images [1]. Combined with better techniques of matching the bag-of-words descriptors, this approach was previously the state-of-the-art [2].

Recently, the state-of-the-art has drastically improved due to the addition of learned features, based on convolutional neural networks (CNNs). This new approach greatly improved mean average precision scores [3] of the system on the same datasets.

The general trend is to move from an approach based on a combination of engineered features (bag-of-words combined with support vector machines or SVMs) to a more end-to-end learning of the matching of two images.

For this, the Siamese architecture is used, where the convolutional features of two or more CNNs with shared weights are combined and a multi-way loss is optimized, which discriminates between the features of two or more images. This concept was first introduced by Chopra et al [4] to learn a dissimilarity metric between two images.

For a learning-to-rank setting, this idea was extended to a triplet loss by Weinberger et al [5]. This loss is better suited when learning to rank, rather than instance search, but it also allows for a more stable convergence, which is beneficial in both cases.

Using this triplet loss has achieved state-of-the-art results in both face recognition [6] and image retrieval [3]. This is mostly due to the usage of far deeper CNNs, moving from architectures such as AlexNet [7] to VGG [8] and finally Inception [9] and ResNet [10].

The higher depth of networks like ResNet and Inception as compared to AlexNet allows for higher regularization and thus less over-fitting to a specific dataset for these architectures. Batch normalization increases this effect even more. This is a desirable trait for image retrieval, as the variability within each instance is too small to learn classification directly. So the better we can generalize features learned from a bigger dataset to the reference dataset, the better performance we can expect.

3 Evaluation

We are using the following datasets in our experiments:

1. CLICIDE: dataset of photographs taken in an art museum, consisting exclusively of paintings. The dataset is characteristic because the different images for each instance consist of one global view of the instance and multiple sub-parts.

   Number of reference images: 3245 (the full dataset contains 3452 images with 207 images of walls and no meaningful instance) Number of test images: 165 (the full dataset contains 177 test images, out of which 165 share their instance with at least one image of the reference set) Number of instances: 464

2. GaRoFou_I: dataset of photographs of an art museum, consisting of display cabinets, which contain sculptures, rocks and various types of objects.

   Number of reference images: 1068 Number of test images: 184 (all are used for evaluation) Number of instances: 311

3.1 Evaluation metrics

3.1.1 Metrics definitions

$Precision@k$ For a test image and m reference images and a ranking of the reference images $I_1, \ldots, I_m$, we define the number of relevant images at k with $k \leq m$: N_k^{rel} is the number of images in the sub-ranking $I_1, \ldots, I_k$ from the same instance than the test image.

We then define Precision at k as:

$$Precision@k = \frac{N_k^{rel}}{k} \tag{1}$$

MAP As above, for a test image, m reference images and a ranking of the reference images, we define the average precision:

$$AP = \frac{1}{N_m^{rel}} \sum_{k=1}^{m} Precision@k \tag{2}$$

The MAP score is defined as the mean of the AP score over all test images.

3.1.2 Used metrics

As of now, we use the mean $Precision@1$ to evaluate the system. It is the mean of the $Precision@1$ for all test images. This is what we are most interested in: in the context of a search system for retrieving art in a museum, we are only interested in one result, as the user should not make a choice out of several results.

To be comparable with other papers in the field, we will implement the MAP score as well later on.

4 Methodology

The following approaches have been tested:

4.1 Features of a pre-trained CNN

We use the features obtained from a CNN that was pre-trained on ImageNet. The feature dimension of the last layer is thus fixed at 1000, the number of classes in ImageNet.

We also use the convolutional features obtained from such a CNN. Their dimensions are specified in the specifics section.

4.1.1 Motivation

This is to set a baseline for other approaches and to reproduce results from a previous paper (TODO cite CORIA paper).

4.1.2 Specifics

All images are resized to 224×224, using a bilinear re-sampling filter. This is to fit the default input dimensions of a CNN used in ImageNet.

The following CNNs were tested:

1. AlexNet (convolutional feature dimension $6 \times 6 \times 256 = 9216$)
2. ResNet-152 (convolutional feature dimension $7 \times 7 \times 2048 = 100352$)

4.2 Features of a fine-tuned CNN

We use the features obtained from a CNN, pre-trained on ImageNet, then fine-tuned as a classification net on the reference dataset.

The feature dimensions are much smaller and correspond to the number of instances of the dataset, described in Section 3.

Similarly to the approach in Section 4.1, we also use the convolutional features obtained from a fine-tuned model, having the same dimensions than described there.

4.2.1 Motivation

The idea is to fine-tune the high-level convolutional filters to our dataset such that they can capture the shapes of that dataset better.

4.2.2 Specifics

All images are resized to 224×224, using a bilinear re-sampling filter.

For fine-tuning the CNN on classification, we use the following settings:

1. Data augmentation: all images are shifted (in range $[-20\%, 20\%]$ in both directions), rotated (in range $[-45, 45]$ degrees), rescaled (with factor in range $[0.8, 1.2]$) and horizontally flipped at random. All parameters are chosen from a uniform distribution.

   The motivation behind the extensive data augmentation is that the training set is very small, so without data augmentation, large CNNs like AlexNet would over-fit the dataset.
2. Number of epochs: 50
3. Learning rate: 0.01 (annealed by a factor of 0.1 after 30 epochs)
4. Momentum: 0.9
5. Weight decay: 0.0005

Note that the hyper-parameters were chosen manually without optimization since they appeared to work well on the CLICIDE dataset. This means they might be far from optimal on both datasets, and especially GaRoFou.

The following CNNs were tested:

1. AlexNet (only last convolutional layer is fine-tuned)
2. ResNet-152 (last 3 blocks are fine-tuned)

4.3 Features of a fine-tuned Siamese CNN using cosine similarity loss

The general principle follows the architecture of Gordo et al [3], omitting the region pooling: We use the convolutional features of a pre-trained CNN. These features are $L2$-normalized, then a shifting layer and fully connected layer are introduced to reduce the feature dimension and the features are $L2$-normalized again. This is similar to the architecture of Schroff et al [6], too.

Most importantly, we use a Siamese architecture along with a cosine similarity loss. The loss is defined as follows for two images and their feature vectors x_1 and x_2:

$$\mathcal{L} = \begin{cases} 1 - cos(x_1, x_2) & \text{if the images are from the same instance} \\ \max(0, cos(x_1, x_2) - margin) & \text{if the images are from different instances} \end{cases} \quad (3)$$

$cos(x_1, x_2) = \frac{x_1 x_2}{\|x_1\|_2 \|x_2\|_2}$. For normalized vectors, we have $cos(x_1, x_2) = x_1 x_2$, so the cosine similarity is simply the dot product here.

4.3.1 Motivation

On small datasets, Siamese architectures should be able to obtain better results than classification architectures as they are fine-tuned to discriminate between two inputs, rather than simply output the class or instance of an input. Because of the missing variability for each instance, a classification architecture will over-fit the dataset more than a Siamese architecture.

As described above, we omit the region pooling layer. There are two reasons behind this: First, we only use images of the same size as inputs, so we do not need to harmonize between different images. Second, the images in our datasets are cleaner than images in the datasets used by Gordo et al (Oxford5k, Paris6k, ...), in the sense that the objects are usually well centered in the images and fill out most of the image. In other datasets, the objects searched for only form a small part of the image. In this case, the rest of the image is noise, making the network more difficult or impossible to train.

4.3.2 Specifics

All images are resized to 224×224, using a bilinear re-sampling filter. The margin in the loss is set to 0.

For fine-tuning the Siamese CNN, we use the following settings:

1. Margin used in loss: $margin := 0$
2. Data augmentation: None/same as in fine-tuning (Section 4.2.2)
3. Batch size: 64/256
4. Number of epochs: 50
5. Learning rate: 0.001
6. Momentum: 0.9
7. Annealing: None

The following choices of the pairs of images for training were tested:

1. Choose all positive pairs and randomly choose negative pairs such that 90% of the training set consists of negative pairs
2. Choose all positive pairs and for each image, choose the 'hardest' x negative pairs: the pairs giving the highest loss. This choice of the hardest pairs is repeated before each epoch during training.

The following CNNs were tested:

1. AlexNet (only last convolutional layer is fine-tuned)
2. ResNet-152 (last 3 blocks are fine-tuned)

4.4 Features of a fine-tuned Siamese CNN using a triplet loss

This resembles the architecture of Gordo et al [3] the most: only region pooling of the convolutional features is omitted.

The loss is defined for an anchor image, a reference positive image and a reference negative image and their feature vectors x_a, x_p, x_n respectively:

$$\mathcal{L} = \frac{1}{2}\max(0, \|x_a - x_p\|_2^2 - \|x_a - x_n\|_2^2 + margin) \tag{4}$$

For normalized feature vectors x_1, x_2, the distance $\|x_1 - x_2\|$ between the vectors is closely related to the cosine similarity: $\|x_1 - x_2\|_2^2 = 2 - 2x_1x_2 = 2 - 2cos(x_1, x_2)$. So the loss could be written as $\max(0, x_ax_n - x_ax_p + margin^\star)$ with $margin^\star = \frac{1}{2}margin$. Both versions of the loss are implemented, but we chose the margin $margin^\star$ as basis since in our case, feature vectors are always normalized.

4.4.1 Motivation

The motivation behind using the triplet loss is laid out in the paper by Weinberger et al [5], who first introduced this loss, as well as the paper by Schroff et al [6]. Essentially, this loss is more robust w.r.t. noise in the data, as we cannot always perfectly project all couples of images of the same instance onto the same point in space, when we have noisy data.

4.4.2 Specifics

All images are resized to 224×224, using a bilinear re-sampling filter. The margin in the loss is set to 0.

For fine-tuning the Siamese CNN, we use the following settings:

1. Margin used in loss: $margin^{\star} := 0.2$
2. Data augmentation: None/same as fine-tuning (Section 4.2.2)
3. Batch size: 64/256
4. Number of epochs: 50
5. Learning rate: 0.001
6. Momentum: 0.9
7. Annealing: None

The following choices of the triplets of images for training were tested:

1. Choose all positive pairs and randomly choose a negative pair for each positive pair to form a triplet.
2. Choose all positive pairs and for each image, choose the 'hardest' negative pair to form a triplet.
3. Choose all positive pairs and for each image, choose 'semi-hard' negatives early on in training, then choose only the 'hardest' negatives. This is motivated by Schroff et al [6].
4. Choose the easiest n_p positives along with the hardest n_n negatives. Then compute the loss for all $n_p \times n_n$ triplets. Choose the n_t triplets with the highest loss. Repeat this procedure until a large enough number of hard triplets are found. Perform this before each epoch during training. This is motivated by the strategy of Gordo et al [3].

The following CNNs were tested:

1. AlexNet (only last convolutional layer is fine-tuned)
2. ResNet-152 (last 3 blocks are fine-tuned)

5 Results

Table 1 shows current results for each approach. The following special values can appear in the table:

- N/A: configuration was not tested
- CH: loss gets smaller, but converges at a high value. usually in this case, testing performance stagnates or goes slightly down. We tried different batch sizes, but none give a better result. This usually indicates that we always converge to a local minimum or the model collapses (setting all embeddings to 0 for example)
- CBT: loss converges towards 0, but testing performance does not improve. this shows the model performs bad somehow

All new results are added there as soon as available.

6 Current work

- Investigate discrepancies between our results and CORIA results. Nothing obvious was found at first sight. However, it seems like the datasets/testing sets are not very stable (because they are very small). This means, for example, that simply choosing a different normalization can have a big impact on results (image-wise normalization vs dataset-wise normalization and in case of dataset-wise, there are different possible ways to do that: pixel-wise . . .)

  Another possibility is the input size of images: we choose the 224×224 size from ImageNet, which gives the expected 6×6 conv. filter maps for AlexNet and 7×7 for ResNet.
- Add visualizations to see whether the filters capture the images as expected: There are several approaches.
  1. Simply display the feature maps of filters as heat maps: this is implemented.

Approach	Dataset			
	CLICIDE		GaRoFou_I	
	AlexNet	ResNet-152	AlexNet	ResNet-152
Full classif. features of pre-trained net	46.06	54.55	76.63	75.54
Conv. features of pre-trained net	44.24	46.67	85.33	84.78
Full classif. features of classif.-fine-tuned net	61.21	75.76	86.96	92.93
Conv. features of classif.-fine-tuned net	52.12	81.21	89.13	96.20
Cosine loss (random pairs) with pre-trained net	N/A	CH	N/A	N/A
Cosine loss (hardest pairs) with pre-trained net	N/A	CH	N/A	N/A
Triplet loss (all pos, random neg) with pre-trained net	N/A	55.15	N/A	N/A
Triplet loss (all pos, semi-hard neg) with pre-trained net	N/A	CH	N/A	N/A
Triplet loss (easy pos, hard neg) with pre-trained net	N/A	CH	N/A	N/A

Table 1: Results in percentage points of mean $Precision@1$ of various approaches on different datasets and underlying networks

2. Calculate the number of times that filters are activated: Are some filters always 0 ? How many times each filter obtains the maximum activation ? This is implemented.
3. Show the gradient for single filter activation: class saliency extraction, not implemented yet
4. optimize response of filter + regularized image by gradient descent (keeping weights fixed): for this and previous approach, see paper by Simonyan et al 'Deep Inside Convolutional Networks..': not implemented yet

7 Future work

Propose new approach based on the problems found from analyzing current systems/approaches, and try to obtain state-of-the-art results on more common datasets, such as Oxford5k, Paris6k or Sculptures. Motivation: current approaches on these datasets as well as the ones presented seem to give far from optimal results / check if our approaches generalize well to other datasets.

- Possibly create new dataset (based on images of museum art pieces)

  Motivation: CLICIDE dataset is particularly suited for ORB. GaRoFou dataset is very small, so the risk of over-fitting to that particular type of dataset is high.

  A better dataset would be one with more images (and proportionally more instances) and a better harmonization of query vs reference images: reference images should be clean and from all angles of the object, query images should not be clean, blurred, only small parts of the image contains object to find . . .

- Possibly investigate few-shots training approaches: the idea is to solve classification with only very few images per class. It does not require pre-trained nets.

References

[1] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[2] A. Mikulik, M. Perdoch, O. Chum, and J. Matas, "Learning Vocabularies over a Fine Quantization," *International Journal of Computer Vision*, vol. 103, pp. 163–175, May 2013.

[3] A. Gordo, J. Almazán, J. Revaud, and D. Larlus, "Deep Image Retrieval: Learning Global Representations for Image Search," in *Computer Vision – ECCV 2016*, pp. 241–257, Springer, Cham, Oct. 2016.

[4] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 539–546 vol. 1, June 2005.

[5] K. Q. Weinberger, J. Blitzer, and L. Saul, "Distance metric learning for large margin nearest neighbor classification," *Advances in neural information processing systems*, vol. 18, p. 1473, 2006.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[8] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Sept. 2014. arXiv: 1409.1556.

[9] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *arXiv:1602.07261 [cs]*, Feb. 2016. arXiv: 1602.07261.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

A Implementation details and specifics

A.1 General issues

large images/large batch sizes/large nets means we require lots of memory this means we implement so-called micro-batches for gradient descent: each mini-batch is decomposed into several micro-batches. perform a forward pass over the micro-batches while accumulating the loss. then, perform a backward pass, which is representative of the mini-batch, but we require only as much memory as for one micro-batch

A.2 PyTorch specific issues

TODO