Deep Metric Learning and Image Classification with Nearest Neighbour Gaussian Kernels (Article Review)

Chervontsev Sergey
Department of Innovation and High Technology
Moscow Institute of Physics and Technology
cherv277@gmail.com

Abstract

Recently, a number of approaches was proposed to use nearest neighbours computation during metric and image classification problems. In this article, I review the recent contributions in this area and propose a few extensions.

1. Introduction

The humans are believed to be extremely biased towards their previous experience. Aside from negative consequences this helps to make learning and inferring more efficient and robust. At the same time, modern deep learning approaches has mostly relied on making isolated predictions. This helps to simplify the training process and makes it feasible even in large-scale settings, but leads to models that are less sample efficient and have higher computation and memory requirements. Thus, there is a particular interest in combining the best of "two words".

The aim of this article is to provide an overview of the few existing works in this area and propose new extensions based on them. The paper review is covered in sections 2 and 3, while sections 4 and 5 describes my proposals.

2. Preliminaries

2.1. Problem Description

Consider the ordinary image classification setting: given a training set $\{(x_i, y_{i,T})\}_{i=1}^N$ that contains images x_i with their one-hot labels $y_{i,T}$ one learns a mapping Y_θ between them. This usually consists of two steps: at first, for a given example x_i we extract its image descriptor z_i. Then, we use a classifier to predict label y_i based on z_i.

For a given model, its quality is estimated based on the Cross-Entropy loss function L_y which we use to get the optimal weights:

$$\theta^* = arg\,min_\theta\{ -\frac{1}{N}\sum_{i=1}^{N} y_{i,T}^T log(y_i)\}. \quad (1)$$

2.2. Feature Extraction

Currently, descriptors are usually extracted using "feature" layers of the popular CNN architectures: VGG [1], ResNet [2], SqueezeNet [3] etc. We would refer to them as "feature extractors" and denote them by f_θ. On top of them, there are multiple versions of classifiers proposed.

2.3. Dense Classifier

In the simplest scenario one may use a single neuron for making predictions, i.e.

$$y(z) = a(Wz + \beta), \quad (2)$$

where $[W, \beta]$ is a pair of learned weights and a is a nonlinear activation function (e.g. softmax). However, if the features dimension is large enough it might be insufficient, and we can switch to multi-layer networks.

2.4. Nearest Neighbours Classifier

As an alternative to linear mapping, one can choose the nearest neighbours classifier which doesn't require any training:

$$y(z) = arg\,max_y|\{z_i|z_i \in \mathcal{N}_k(z) \;\&\; y_i = y\}|, \quad (3)$$

where $\mathcal{N}_k(z)$ is a set of k closest points z_i for z according to Euclidean metric.

2.5. Kernel Neuron Classifier

The linear classifier doesn't account for proximity of the data points, while the nearest neighbours classifier is too simple. A kernel neuron [4] provides a trade-of between

them with the following computation scheme:

$$y(z) = a(\sum_{i=1}^{N} \alpha_i k(z, c_i) + \beta). \quad (4)$$

Here, α_i are sample-importance weights, β is a shared bias term, c_i are learned centers and $k(z, c_i)$ is a kernel function that measures similarity between z and c_i. Generally, one uses a Gaussian (RBF) kernel:

$$k(z, c_i) = exp(\frac{-\|z - c_i\|_2^2}{2\sigma^2}). \quad (5)$$

Note that $k(z, c_i)$ quickly goes to zero as the distance between z and c_i approaches infinity. Therefore, on large-scale datasets, in equation (4) we can account only for the nearest neighbours of z:

$$y(z) = a(\sum_{i \in \mathcal{N}_k(z)} \alpha_i k(z, c_i) + \beta). \quad (6)$$

3. Nearest Neighbours Gaussian Kernels

Motivated by the kernel neuron model (4), the authors of [5] proposed a scalable and feasible approach of the embedding learning for image classification and distance metric tasks.

Their main contribution is to account for nearest neighbours in computation, as described in (6). For a particular z, those could be extracted fairly quickly using the Fast Approximate Nearest Neighbours Graph [6]. Compared with (6), the center positions c_i are chosen to be equal to the current image descriptors $z_i = f_\theta(x_i)$. This leads to computation issues, as that means c_i for every i must be updated after each gradient step. However, the conducted experiments showed that the "weak" update of c_i (for example, after a few epochs of training) is sufficient if we increase the amount of considered neighbours up to 32(see [5]).

Additionally, the bias is discarded and the normalization activation is chosen, which leads to the following formula:

$$y(z) = \frac{\sum_{i \in \mathcal{N}_k(z)} w_i y_i k(z, z_i)}{\sum_{i \in \mathcal{N}_k(z)} w_i k(z, z_i)}. \quad (7)$$

The model is optimized by performing gradient descent on (1). For simplicity, it doesn't involves optimizing z_i but only z. Intuitively, this can be understood as "nudging" the descriptor z towards the neighbours of the same class y while pushing it away from the neighbours that have a different class.

3.1. Experiments

The original paper proposed a number of experiments to demonstrate the power of their model.

Firstly, they evaluate the embedding results on Stanford Cars [7] and Caltech Birds [8] datasets. The use of Nearest Neighbours Gaussian Kernels improves the Recall@K metric by a perceptible margins, especially in case of low K-s. E.g. for Cars196, the model can reach a Recall@4 metric of 92.36, while the second best model gets only 84.23. On Birds200, even the Recall@8 gets an improvement of approximately 3.8, from 83.31 to 87.17.

Secondly, they computed the effect of the number of training examples per class on the test set accuracy of Birds200 dataset for fine-tuning VGG16 architecture, and compared it with a classic softmax classifier. They found that the model can provide a great improvement when this number is low, e.g. approximately 10 percents (from 49 to 60) in case of a six samples per class.

3.2. Extensions

The main subsequent work from this paper is the one by Lebedev et al [9]. There, the authors consider the model described in (7) and introduce multiple extensions for it.

The first extension is to re-introduce the training of Gaussian centers c_i. To prevent overfitting, they are forced to "stay close" to the tied descriptors z_i by adding L2 regularization term. Thus, one may call them "impostors" as they mimic the "true" centers z_i. The second extension is to compress the center representation by applying Optimized Product Quantization [10]. Additionally, they drop the sample-importance weights w_i.

Overall, the proposed extensions lead to the following training objective:

$$L(\theta, c) = L_y(\theta, c) + \frac{\lambda}{N} \sum_{i=1}^{N} \|f_\theta(x_i) - c_i\|_2^2, \quad (8)$$

where $L_y(\theta, c)$ is a common Cross-Entropy loss as described in (1), and the predictions are computed as follows:

$$y(z) = \frac{\sum_{i \in \mathcal{N}_k(z)} y_i k(z, c_i)}{\sum_{i \in \mathcal{N}_k(z)} k(z, c_i)}. \quad (9)$$

Additionally, they tried two different ways for selecting centers: tied impostors by constraining $c_i = f_\theta(x_i)$ as in (7) and fixed impostors by choosing $c_i = f_{\theta_0}(x_i)$. However, the "loose" impostors scheme described in (8) often gives the best accuracy and never performs much worse(see [9] for details).

3.3. Pros and Cons

Overall, the presented Nearest Neighbours classifier offers a number of interesting properties compared with the traditional approach. It accounts for relative distance between image descriptors and thus can be successfully used for ranking even in the case of being trained for classification tasks. Also, with the help of ANNG search, it offers

a faster inference even in the case of considering lots of neighbours (e.g. a hundred).

Additionally, despite the big numbers of training samples N, it is possible to store all centers c_i even in large dimensions using relatively small memory. This is clearly shown in [9], where the authors claim that for 5994 training samples on Birds200 with descriptors having 512 dimensions, it requires only 92 KB to store the 16 bytes centers, while the size of the SqueezeNet feature extractor they use is 4.8 MB.

Probably, the main disadvantage of this architecture is the increased complexity of the model, as one should use external tools for fast neighbours search. It's also questionable whether one needs to use it on large-scale datasets. E.g. in the case of ImageNet, one would require $\frac{1,4e7}{6e3} \approx 2300$ times more space for storing centers. However, I believe those cons are minor compared with the pros it offers.

4. Proposed Research: NNGK for Continual Learning

4.1. Problem Description

In the continual learning setting, a training set we consider is $D = \{(x_i, t_i, y_{i,T})\}_{i=1}^N$, where t_i is a task descriptor, and $y_{i,T}$ is a label for this particular task. The simplest case is the ordinary classification task where the training set is slightly changed after each a few training epochs. Traditional networks trained with empirical risk minimization scheme are known to suffer from catastrophic forgetting, meaning that they quickly forget previous learned samples to optimize better on new.

Recently, a number of solutions was proposed to alleviate this problem, including elastic weight consolidation [11], gradient episodic memory [12], learning through the intelligent synapses [13] and conceptor-aided backpropagation [14]. I believe them to be over-complicated and propose a different approach, based on the described model.

4.2. Model

I consider the standard Impostor Network, as described in 3.2. Note that in equations (8) and (9), there is no need to keep specifically N centers. I expect we can drop some of the centers once we see their labels are well predicted by their neighbourhood. In the same time, for a new training pair (z_n, y_n) which is hard to predict, we may add z_n in our centers set to improve the results in its neighbourhood.

However, as we update our centers set, the predictions on non-updated points also changes, which may increase our loss significantly, thus compensating all of the benefits we get from updating. To tackle this issue, I propose to redistribute the targets for the old centers. More specifically, let us consider we change (either drop or add) the training sample (x_0, z_0, y_0). Let us define the following "influence" function for arbitrary descriptor z:

$$I_0(z) \triangleq \frac{\partial y_0(z, \varepsilon)}{\partial \varepsilon}\Big|_{\varepsilon=0}, \tag{10}$$

where the l-th component of $y_0(z, \varepsilon)$ is:

$$y_0^l(z, \varepsilon) = \frac{\sum_i y_i^l k(z, z_i) + \varepsilon y_0^l k(z, z_0)}{\sum_i k(z, z_i) + \varepsilon k(z, z_0)}. \tag{11}$$

By substituting (11) in (10), we get:

$$I_0^l(z) = \frac{y_0^l k(z, z_0) \sum_i k(z, z_i) - k(z, z_0) \sum_i y_i^l k(z, z_i)}{(\sum_i k(z, z_i))^2}. \tag{12}$$

Now for a particular training point z_j, if we take enough neighbours, it's likely that $\frac{k(z_j, z_0)}{\sum_i k(z_j, z_i)} \ll 1$. Therefore, the change of the label at this sample can be estimated as:

$$\Delta y_j \approx I_0(z_j) = \frac{k(z_j, z_0)}{\sum_{i \in \mathcal{N}_k(z_j)} k(z_j, z_i)}(y_0 - y_j). \tag{13}$$

And can be compensated manually for each point in the neighbourhood of z_0. Overall, this leads me to the following procedure:

Result: A trained model M_θ
$D \leftarrow \{D_e\}_{e=1}^E$— a set of changing datasets;
$C \leftarrow \{f_{\theta_0}(x_i)\}_{i=1}^N$— a set of active centers;
$n_d, n_a \leftarrow$ number of centers to drop and add;
$n_t, n_f \leftarrow$ number of training and fine-tuning epochs;
for $e = 1, \ldots, E$ *"mega" epochs* **do**
  Train the current model on D_e for n_t epochs;
  Freeze the model weights;
  Collect n_d best predicted samples from C in S_d and n_a worst predicted samples from $D_e \setminus C$ in S_a;
  for $c_0 \in S_d$ **do**
    Drop the center c_0 and re-distribute the label y_0 across its neighbours using (13);
  end
  Same with S_a;
  Unfreeze model and continue training for n_f epochs
end

Algorithm 1: Continual learning with Impostor Network

It may help to solve continual learning scenario as in the simplest case, we may fix $n_d = 0$ and only add the most ambiguous descriptors z_i to the centers set.

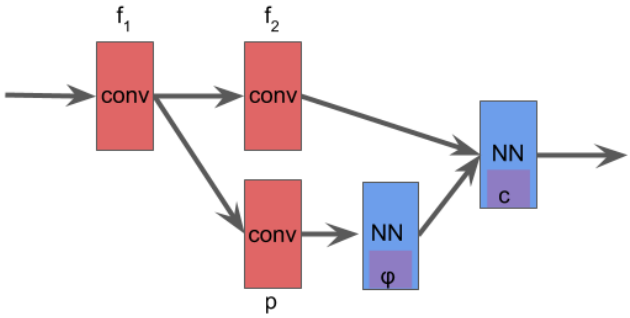

Figure 1. A computational scheme for two-layer NNGK Network. "conv" denotes classical pooling/convolution block, while "NN" denotes Nearest Neighbours Gaussian Kernel classifier. f_1 and f_2 are two steps of feature extraction, while p is a projector network.

4.3. Experiments

To check the validity of this model, we may start with a convenient Split CIFAR-10/CIFAR-100 [15] benchmark, as described in [13]. The baselines could be a standard CNN architecture and the Impostor Network. In case of success on this task we can move on to more complicated datasets, such as CORe50 [16] or Atari Games [17].

5. Proposed Research: Biased Network

5.1. Problem Description

As described in 3.3, in case of Impostor network with a SqueezeNet feature extractor, both computational and memory requirements of classifier are significantly less than those for CNN. It is thus natural to wonder whether some of the benefits of Fast Approximate Neighbours search can be utilized inside the feature extractor.

In this section, I speculate on how can we boost the computation time on a particular examples, based on the reviewed model. Potentially, this may not only speed up performance, but also give us more adversarially robust models. The latest research in this area shows that efficient adversarial defence may be strongly correlated with the ability to concentrate computational power in the proximity of the "true" images manifold, while performer simpler outside of it. E.g. Verma et al [18] show that if we train the model on samples that contains linear combinations of images with the same combination of labels, we get a model that is more stable to one-step attacks.

5.2. Model

Due to improved computational and memory requirements, one may want to simply stack multiple layers of (7). However, this approach faces multiple issues. Firstly, as the dimension of descriptors increase, so does the memory requirements for the centers set. Secondly, the NNGK is a relatively simple model - it will be probably insufficient in its predictions until the data points could be unfolded in a less complicated manifold. Lastly, in the spaces of large dimension, the distance function faces the curse of dimensionality. Thus, one may require to have too many samples in the training set.

To overcome these problems, I propose to use a separate "projector" network. The computational scheme of the proposed model can be described as follows:

$$
\begin{aligned}
h &= f_1(x), \\
\phi &= p(x), \\
z_1 &= \frac{\sum_{i \in \mathcal{N}_k(\phi)} z_i k(\phi, \phi_i)}{\sum_{i \in \mathcal{N}_k(\phi)} k(\phi, \phi_i)}, \\
z_2 &= f_2(x), \\
g &= \frac{1}{|\mathcal{N}_k(\phi)|} \sum_{i \in \mathcal{N}_k(\phi)} k(\phi, \phi_i), \\
z &= g z_1 + (1 - g) z_2, \\
y &= \mathrm{clf}(z).
\end{aligned}
\tag{14}
$$

In essence, the idea is to mix traditional convolution computation with nearest neighbours in case we encounter them. That's it, for a particular image x we get its low-layer representation h. Then, we simultaneously compute its "true" descriptor z_1 and its "biased" version z_2. To get z_2, we use a weak projector p which gives us ϕ in lower dimension and after this, use a NNGK layer with stored centers ϕ_i, which are tied with the representations of z_i. Finally, we apply a gating mechanism with g, that measures whether this projected representation was close to its neighbours or not and thus, how we may trust in z_1.

Note that, compared to f_2 the projector p doesn't have to separate the points perfectly: in worst case, it can push all centers ϕ_i far away from the manifold of observed features ϕ, and the model predictions will completely rely on

f_2. However, as ϕ_i are initialized using the projector over the true images, I believe the accurate choice of p can prevent the complete degradation. Additionally, this scheme provides us with the way to partially boost computations - during inference we may firstly compute z_1. If the gate value g is close to one, we may omit the computation of z_2, assuming $z \approx z_1$.

5.3. Experiments

The organization of Inception network make it a good starting point for the proposed model, as we can remove the last layers in its auxiliary classifiers and use them as projectors. As a proof of concept, we may take moderate but competitive datasets, like CIFAR100 or TinyImageNet. Additionally, we need to check in how many samples may we drop the computation of z_2 and whether the most "unexpected" inputs make any sense. We may also check the model robustness against the black-box attacks.

References

[1] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[2] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[3] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size". In: *CoRR* abs/1602.07360 (2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360.

[4] YandaLi Jianhua XU Xuegong Zhang. "Kernel Neuron and Its Training Algorithm". In: *8th International conference on neural information*. Vol. 2. 2001, pp. 861–866. URL: /papers/upload%5C_8320%5C_Kernel%5C_Neuron%5C_ICONIP2001.pdf.

[5] Benjamin J. Meyer, Ben Harwood, and Tom Drummond. "Nearest Neighbour Radial Basis Function Solvers for Deep Neural Networks". In: *CoRR* abs/1705.09780 (2017). arXiv: 1705.09780. URL: http://arxiv.org/abs/1705.09780.

[6] Ben Harwood and Tom Drummond. "FANNG: Fast Approximate Nearest Neighbour Graphs". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 5713–5722.

[7] Jonathan Krause et al. "3D Object Representations for Fine-Grained Categorization". In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[8] P. Welinder et al. *Caltech-UCSD Birds 200*. Tech. rep. CNS-TR-2010-001. California Institute of Technology, 2010.

[9] Vadim Lebedev, Artem Babenko, and Victor S. Lempitsky. "Impostor Networks for Fast Fine-Grained Recognition". In: *CoRR* abs/1806.05217 (2018).

[10] Tiezheng Ge et al. "Optimized Product Quantization". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 36.4 (Apr. 2014), pp. 744–755. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.240. URL: https://doi.org/10.1109/TPAMI.2013.240.

[11] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *CoRR* abs/1612.00796 (2016). arXiv: 1612.00796. URL: http://arxiv.org/abs/1612.00796.

[12] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient Episodic Memory for Continuum Learning". In: *CoRR* abs/1706.08840 (2017). arXiv: 1706.08840. URL: http://arxiv.org/abs/1706.08840.

[13] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Improved multitask learning through synaptic intelligence". In: *CoRR* abs/1703.04200 (2017). arXiv: 1703.04200. URL: http://arxiv.org/abs/1703.04200.

[14] Xu He and Herbert Jaeger. "Overcoming Catastrophic Interference by Conceptors". In: *CoRR* abs/1707.04853 (2017). arXiv: 1707.04853. URL: http://arxiv.org/abs/1707.04853.

[15] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009.

[16] Vincenzo Lomonaco and Davide Maltoni. "CORe50: a New Dataset and Benchmark for Continuous Object Recognition". In: *CoRR* abs/1705.03550 (2017). arXiv: 1705.03550. URL: http://arxiv.org/abs/1705.03550.

[17] Vitaly Kurin et al. "The Atari Grand Challenge Dataset". In: *CoRR* abs/1705.10998 (2017). arXiv: 1705.10998. URL: http://arxiv.org/abs/1705.10998.

[18] Vikas Verma et al. "Manifold Mixup: Learning Better Representations by Interpolating Hidden States". In: *stat* 1050 (2018), p. 4.