Keypoints selection using Evolutionary Algorithms

David Adamczyk, Jan Hůla

Institute for Research and Applications of Fuzzy Modeling
david.adamczyk@osu.cz
jan.hula@osu.cz
University of Ostrava, 30. dubna 22, 701 03 Ostrava, Czech Republic

Abstract: This contribution presents the use of neural networks trained by an evolutionary algorithm for a selection of visual keypoints. Visual keypoints play an important role in many computer vision tasks but many algorithms for keypoint detection produce many keypoints which are not useful for the target task. We aim to filter them in a data-driven way. Our model uses a neural network that ranks each keypoint by a relevancy score that we use to choose top-K keypoints with the highest rank. These keypoints are then used for the target task, which is image classification in our case. Because we use discrete operations in our model, we can not easily obtain gradients for weight updates. We, therefore, optimize the weights of the network by CMA-ES algorithm, which enables efficient optimization of continuous parameters of black-box functions. In this article, we present our initial experiments with this method.

1 Introduction

In many problems of machine learning, we deal with sets of features that describe objects we want to process. In some cases, these sets are ordered, like, for example, in tabular datasets where each type of feature corresponds to one column. In other cases, they are unordered, like, for example, with visual keypoint descriptors that are extracted by keypoint detection algorithm. Feature selection is a well known preprocessing step for the case where these features are ordered. In the case of tabular datasets, we may, for example, filter out specific columns of the table which contain redundant or unrelated information. Feature selection is much harder for unordered sets of features because we need to figure out which features to throw out for each example separately based on the content of these features. Here we present an approach for feature selection of unordered sets where each feature is ranked by a neural network whose weights are learned by a variant of evolutionary algorithm called Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We showcase it for the problem of visual keypoint selection, which could be viewed as a preprocessing step for many computer vision algorithms.

2 The role of Keypoints in Computer Vision

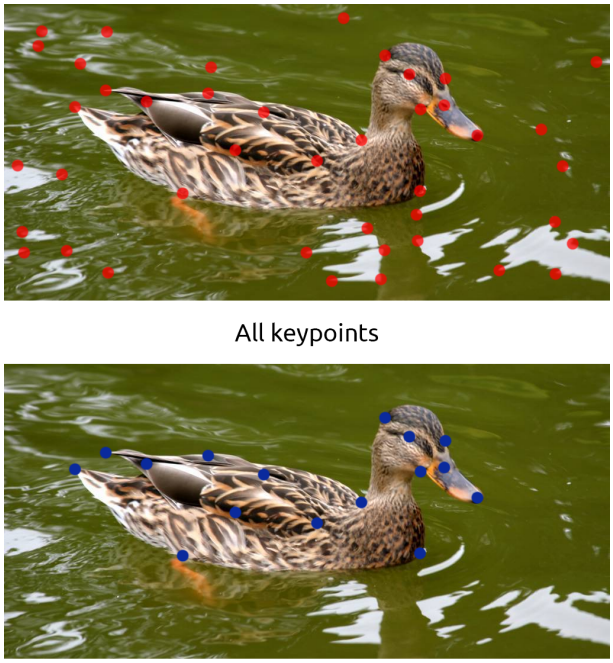

Figure 1: An illustration of the keypoint selection process. Our algorithm learns to select keypoints which are relevant for image classification.

Keypoints, sometimes also called points of interest or visual keypoints, are points in an image represented by coordinates of pixels. They are usually used to extract local feature descriptors that provide a representation of an image that is invariant with respect to transformations such as translation, rotation, scale, or other affine transformations.

There are many well-known algorithms for keypoint detection. Many of them are based on edge detection, corner detection, or blob detection. The best-known algorithms are probably SIFT (Scale-invariant feature transform) [1] or SURF (Speeded up robust features) [2]. In the SIFT algorithm, the keypoint locations are defined as maxima and minima of the result of the difference of Gaussians applied in scale space to a series of smoothed and resampled images. SURF algorithm is based on square-shaped filters as an approximation of Gaussian smoothing.

Keypoints are especially useful for tasks such as image registration, where we want to transform multiple images into one coordinate system or 3D reconstruction, where we want to find 3D coordinates of each pixel. In both of these tasks, we need to match keypoints from two or more images by a matching algorithm. We assume that the keypoints we are trying to match come from images capturing the same scenery but from different viewpoints or lighting conditions. Therefore all images we are trying to match should contain similar keypoint descriptors. In other tasks, such as image classification, this assumption may no longer hold. In image classification, we may try to classify each image by matching its keypoint descriptors to keypoint descriptors of prototype images of different classes. We would assign the image to a class of the prototype image with the best match. The problem arises when images of objects in the same class contain, for example, different backgrounds, which will be the case most of the time. Keypoints detected outside of the object may not be correlated with the class, and this may decrease the accuracy of the classification. For this reason, keypoints are not very often used for tasks such as image classification. This could be changed if we can filter out keypoints that are not relevant for the task at hand, which is a problem we address in this contribution.

3 Optimization of non-differentiable objective functions with continuous parameters

Deep Neural Networks, which are currently the most popular approach in Machine Learning and Computer Vision, are most of the time trained by variants of gradient descent algorithm that require differentiable objective function. The objective function itself is a proxy for the metric we really care about. For example, in classification, we mostly care about classification accuracy, but we use an objective function such as cross-entropy for training because accuracy is not differentiable, but cross-entropy is. For optimization problems in which the model makes discrete choices during processing, the objective function will not be differentiable, and therefore such problems are often approached with Reinforcement Learning where gradients are estimated by various gradient estimation techniques or by Evolutionary Algorithms which do not use gradients but perform a kind of random search. In our case, we need to optimize the parameters of a neural network, which will rank each keypoint by a relevancy score. We choose top-K keypoints with the highest relevancy score, and this discrete choice makes our objective function non-differentiable. Our choice of the optimization algorithm is CMA-ES, which is described in the next section.

4 CMA-ES

Evolutionary Strategies represent a subclass of optimization algorithms inspired by natural selection. They are stochastic, derivative-free methods aimed for numerical optimization of non-linear or non-convex continuous optimization problems. They search through the space of continuous parameters by using a mutation and selection operators, which are interleaved in an iterative process. One iteration (mutation and selection) of this process corresponds to one generation. In Evolution Strategies, the mutation operator produces each individual by sampling from a multivariate gaussian distribution with mean μ and covariance matrix Σ. The Selection operator selects n best individuals from the current generation using the function f we are trying to optimize. We treat this function as a black box. Therefore we can evaluate it for an individual $\mathbf{x}$ to get its fitness $f(\mathbf{x})$, but otherwise, we have no assumptions about it. The next generation is sampled from a multivariate gaussian distribution with parameters depending on selected individuals from the previous generation. Here are the steps of the algorithm for generic distribution parametrized by θ:

1. Generate population of individuals $\Lambda = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ where $\mathbf{x_i} \sim p_\theta(\mathbf{x})$.
2. Compute a fitness score $f(\mathbf{x})$ for all individuals in Λ.
3. Select subset of individuals with the best score. Based on this subset, update the value of parameter θ.
4. Repeat.

In all types of Evolution Strategies, we adapt the mean μ for the gaussian distribution. In CMA-ES we also adapt the covariance matrix Σ. This allows the algorithm to adapt the step size for each dimension of the parameter vector $\mathbf{x}$ separately. Adaptation of the covariance matrix can be seen as an estimation of a second-order model of the underlying function f. It is similar to the approximation of the inverse Hessian matrix in the quasi-Newton method. For the concrete form of the update equations for μ and Σ see the tutorial in [3].

5 Model and task description

Our task was inspired by [4] where a Reinforcement Learning agent is trained to play a game from a screen of pixels. The agent uses an attention module which selects positions in the screen where the agent should pay its attention and filters out the rest of the screen. Authors of this method showed that the agent learned to pay attention to objects/positions, which are essential for the game. We adapt attention module described in [4] together with the learning algorithm (CMA-ES) and apply it to the task

of keypoint selection, which we believe could have a high practical impact.

We assume that we are given a set of N keypoints in an image extracted from some keypoint detection algorithm and we want to choose a subset of K keypoints (with $K \ll N$) that will contain a useful information for the task at hand.

The task we consider here is image classification. For each class, we assume R prototype images[1] which are representative examples of the class. Each image is represented by a set of keypoints. For a new image, we can match its keypoints to keypoints of prototype image p and obtain a matching score. To match the keypoints, we solve a linear assignment problem where the cost for every pair of keypoints is computed by a distance between their keypoint descriptors. The matching score is computed by summing the distances between matched keypoints. After averaging the matching score over R prototype images in each class, we can assign the new image to the class with the highest average score.

Given the type of distance function used for each matching pair of keypoints[2], the average matching score depends only on the sets of keypoints used for matching. Therefore, if we have a training dataset $X = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M))$ of images $\mathbf{x}_i \in \mathbb{R}^{W \times H}$ with labels $y_i \in 1, \ldots, P$, where W, H, P are width and height of an image and a number of classes respectively, we can optimize a function f_ϕ which selects a subset of K keypoints from the set of J keypoints, based on a criterion c which measures how many images were assigned to the correct class. We describe the optimization criterion c later in this section.

The function $f_\phi : \mathbb{R}^{N \times D} \to \mathbb{R}^{K \times D}$ gets as an input a set of N real-valued vectors (keypoint descriptors) of dimension D and returns a subset of K elements from it. It is the attention module which we adapted from [4]. Here we provide a pseudo-code implementation of this function with an informal description below.

1: **function** GET-TOP-K(A, ϕ)
2: $F_1, F_2 \leftarrow \text{INIT}(\phi)$
3: $Q \leftarrow A \cdot F_1$
4: $S \leftarrow A \cdot F_2$
5: $P \leftarrow S^T \cdot Q$
6: $V \leftarrow \text{COLUMN-WISE-SOFTMAX}(P)$
7: $z_1 \leftarrow \sum_j V_{ij}$
8: $z_2 \leftarrow \text{ARGSORT}(z_1)$
9: $ixs \leftarrow z_2[0:K]$
10: $S \leftarrow A[ixs]$
11: **return** S
12: **end function**

The input to the function is a matrix $A \in \mathbb{R}^{N \times D}$ with one keypoint descriptor per row and a vector of learnable parameters ϕ. On **line 2** we initialize matrices $F_1, F_2 \in \mathbb{R}^{D \times \Xi}$ where Ξ is a hyperparameter[3]. These matrices can be seen as weights of two 1-layer linear neural networks. The parameter vector ϕ has a dimension $D \cdot \Xi + D \cdot \Xi$ and the function INIT only splits this vector in half and reshapes the two parts to a matrix. On **lines 3 and 4** we create 2 low-dimensional representations of each keypoint descriptor and store them to matrices Q and S. On **line 5** we compute inner products between these two low-dimensional representations for each pair of keypoints. For a better intuition, the inner product between Q-representation of keypoint k_1 and W-representation of keypoint k_2 can be seen as a vote that k_1 is giving to k_2. On **line 6** we normalize these inner products for each column of matrix P by a softmax function so that the whole column sums to 1. This can be seen as a restriction for the keypoint k_1 to distribute its one vote to all other keypoints. On **line 7** we compute the score of each keypoint k_i by summing the votes from every other keypoint k_j. On **line 8** we sort the indices of keypoints by their score. On **lines 9 and 10** we choose top-K indices and select keypoint descriptors that belong to these indices from a matrix A. These are then returned from the function.

We optimize the parameters ϕ with respect to the following criterion c:

$$c(\phi) = \sum_{i=1}^{M} \mathbb{1}(class(f_\phi(\mathbf{x_i})), y_i),$$

where $\mathbb{1}$ is an indicator function and $class : \mathbb{R}^{K \times D} \to \mathbb{N}$ is the function which assigns the class to an image based on the selected keypoints as described above. Therefore we have an optimization problem in the form of:

$$\phi = \arg\max_\phi c(\phi).$$

As in [4], we approach it with CMA-ES algorithm.

6 Experiments

We conduct two types of experiments that test the viability of our method. The first experiment was conducted on a synthetically generated dataset, where the hardness of the problem could have been controlled manually. The second set of experiments was conducted with a realistic dataset called Willow-Objects [5].

6.1 Synthetic dataset with gaussian feature descriptors

This experiment enabled us to start from an easy case where we expected the algorithm to work and then incrementally make it harder by either making feature descriptors noisier or adding distracting keypoints. Here we describe the final form of the dataset for which we show results in the Table 1.

The dataset contains 10 classes with 1500 examples per class, where each example is represented by 20 feature descriptors. Each feature descriptor is in turn a 200-dimensional vector. Therefore the dataset has a form $X =$

[1] In our case $R \in \{5, 10\}$.
[2] In our experiments, we use cosine and euclidean distance.
[3] In our case it is 10.

$\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_M}, y_M)\}$ where $M = 15000$, $\mathbf{x_i} \in \mathbb{R}^{20\times 200}$ and $y_i \in \{0, \ldots, 9\}$.

We wanted to model the fact that some classes could share the same types of feature descriptors. Each type of feature descriptor is represented by a mean $\mu_\mathbf{j}$ and covariance matrix Σ_j. Particular feature descriptors are then sampled from Gaussians with these parameters. The type of feature descriptor could be viewed as an abstract object, such as "eye," and the particular sample can be viewed as a particular image of an eye. We want some types of feature descriptors to be unique to a class and others to be shared by more classes. For example, an eye could be a useful type of feature descriptor, which is shared by classes such as "DOG" or "CAT," but it is not present in classes such as "CAR" or "APPLE." To model this, we represent classes as leaves in a binary tree where classes that share more common ancestors will share more common types of feature descriptors. Every class/leaf is assigned six unique feature descriptors, and every common ancestor adds one shared type of feature descriptor. To get ten classes, we sample ten random paths in a binary tree, which is five levels deep. This will produce ten types of feature descriptors (or ten indices j indexing the parameters $\mu_\mathbf{j}$ and Σ_j) per class. These types of feature descriptors contain information that is relevant for classification. Every example $\mathbf{x_j}$ from the same class c will contain samples from 10 Gaussians, which correspond to the class c.

We also want to model distracting feature descriptors that would correspond to background clutter. For these, we reserve 60 unique types of feature descriptors represented by 60 new values for the index j. For each example $\mathbf{x_i}$, we first sample 10 indices from these 60 reserved and then sample random vectors from distributions corresponding to these indices. Together, each example in the class c is represented by 20 200-dimensional vectors. Ten vectors are always sampled from the same distributions corresponding to class c, and ten vectors are sampled from ten randomly chosen distributions from the pool of 60, for each new example separately.

The parameters of each gaussian corresponding to one type of feature descriptor are sampled randomly, but in such a way that the gaussian ellipsoids corresponding to different types of feature descriptors are well separable in the 200-dimensional space. To achieve this, we set the means of these Gaussians to coordinates of corners of the unit 200-simplex. Concretely, the mean $\mu_\mathbf{j}$ will have 1 on position j and zeros everywhere else. The covariance matrix is sampled from ranges that guarantee that the samples are well separable. In order to make the task more difficult, we process each sample with a randomly initialized 2-layer neural network, which preserves the dimension. This will nonlinearly deform the space, and also, the resulting clusters will not be axis-aligned.

Our model is trained to select 10 out of 20 feature descriptors that are relevant for classification. Because we knew exactly which feature descriptors contain the relevant information, this dataset allowed us to develop the algorithm with more confidence, without worrying whether problems arise due to the algorithm or due to the dataset. We split the whole dataset to 10000 training examples and 4900 testing examples. The remaining 100 examples are used as prototypes to which we match the selected feature descriptors (10 prototypes per class). After we achieved 85% accuracy on the test set, we moved to experiments with realistic examples.

6.2 Experiments on Willow-Object dataset

Our next set of experiments was conducted with the dataset called Willow-Object [5]. We choose this dataset because it contains annotated keypoints, which we could use for the debugging of the algorithm. The dataset contains 5 classes, each with 40 example images. To obtain feature descriptors for an image, we use a keypoint detector in the SIFT algorithm to obtain 400 keypionts per image. For each keypoint in the image, we extract a vector of activation values from two layers of a pre-trained neural network in the spatial position corresponding to that keypoint. Concretely, we use VGG-11 pre-trained on ImageNet and layers named **relu4_2** and **relu5_1** . For each keypoint we obtain a 1024-dimensional vector. Therefore, each example $\mathbf{x_i} \in \mathbb{R}^{400\times 1024}$ is represented by 400 1024-dimensional vectors from which we wanted to select 10 vectors used for matching.

In this experiment, we found that cosine distance produces much better matches than the euclidean one. We figured this out, with the help of manually annotated keypoints in this dataset. Each image contains 10 annotated keypoints which correspond to parts of the object. For example, each image in the class "FACE" will contain keypoints for eyes, nose, mouth, etc. Together, there are $5 \times 10 = 50$ types of different keypoints. We can, therefore, measure an average distance between each pair of keypoint types, by computing distances of all possible pairs from two types of keypoints, e.g., "EYE" and "WHEEL", and averaging them. Naturally, we would like the average distance to be smallest between keypoints of the same type so that the matcher is encouraged to match keypoints of the same type together. We found that this property holds for cosine distance, but not for euclidean distance, as shown in Figure 3. We speculate that this may be due to the fact that the keypoint descriptors extracted from a pre-trained network may be rather sparse because they are activation values of the ReLU function and for such descriptors, cosine distance may work better.

We split the dataset in such a way that for each class, we have 30 training examples, five prototype examples, and five test examples. As shown in Table 1, we achieved perfect accuracy on the train and the test set. The selected keypoints are shown in Figure 2.

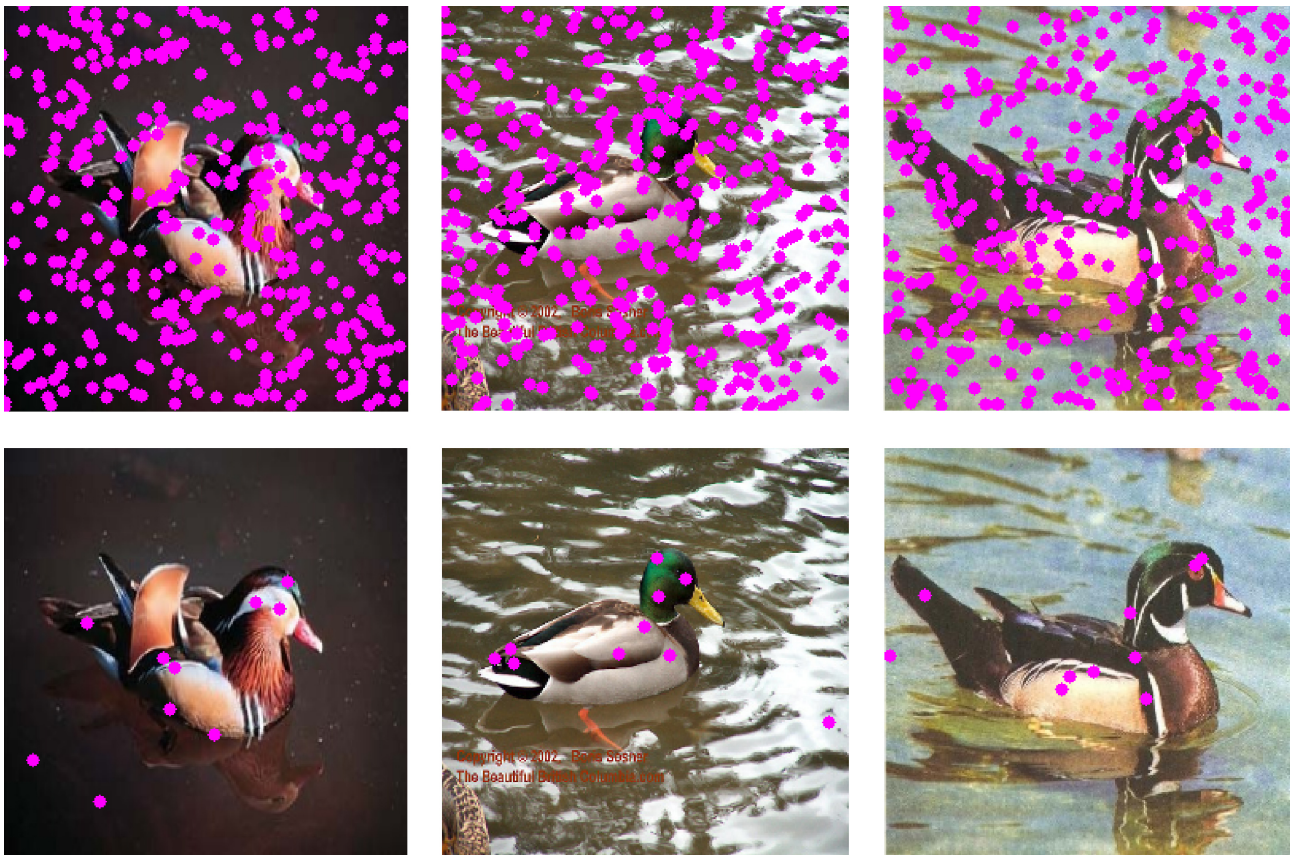

Figure 2: Examples of selected keypoints. Top row: keypoints from SIFT keypoint detector, Bottom row: 10 keypoints selected by our algorithm.

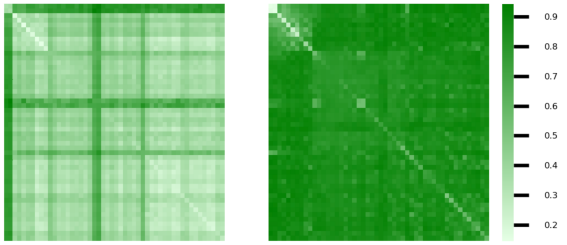

Figure 3: A visualization of a distance matrix for all pairs of keypoint types. An element on row i and column j corresponds to an average distance between all keypoints of type i and all keypoints of type j. Cosine distance (RIGHT) works much better than euclidean distance (LEFT) for matching keypoints of the same type together.

	distance	Train acc.	Test acc.
Synthetic dataset	euclidean	0.86	0.85
Willow-Object-SIFT	cosine	-	0.74
Willow-Object-SIFT	euclidean	-	0.34
Willow-Object-filtered	cosine	1.0	1.0

Table 1: Training and testing accuracy for our experiments. Willow-Object-SIFT uses all keypoints extracted by SIFT. Willow-Object-filtered uses keypoints selected by our algorithm.

7 Related work

The approach described in this work is based on techniques used in [4], which uses parallel selective attention to identify essential parts of the environment. This kind of parallel attention mechanism was popularized by [6]. Previously approaches modeling attention [7] were sequential in nature, in Computer Vision, for example, mimicking saccadic eye movements. In most cases, the attention is differentiable and modeled by a softmax distribution over possible positions [8]. This kind of attention is called soft attention as opposed to hard attention [9–11], where discrete choices are made, usually with a maximum operator. Whereas soft attention modules can be trained with gradient descent, hard attention modules were usually trained with Reinforcement Learning. As far as we know, [4] were the first one to use CMA-ES to train a parallel hard attention module. We tried to adapt their method to keypoint filtering, which we believe can have a more immediate practical impact. Our contribution is also related to other articles about keypoint selection and discovery. In [12] he authors described a method named Iterative Keypoint Se-

lection, where the main idea is to select representative keypoints in the first step and then filter them using a distance-based rule. Keypoints for which the distance is higher than a predefined threshold are removed in an iterative fashion.

The Transporter [13] is a neural network architecture for unsupervised keypoint discovery from video frames. The discovered method enables two notable results in control domains. Using the keypoint co-ordinates and corresponding image features as input enables highly sample-efficient reinforcement learning, and learning to explore by controlling keypoint locations drastically reduces the search space. Another architecture named KeypointNet [14] is used for the detection and discovery of 3D keypoints from 2D images. Their model discovers geometrically and semantically consistent keypoints across viewing angles and instances of an object category.

8 Discussion

In our experiments, we worked with datasets that contained annotated keypoints in order to test our hypothesis that the algorithm will be able to recover the relevant keypoints that we knew were present. These datasets enabled easier debugging and development of the algorithm. In the future, we will continue our work with more realistic datasets, which contain more training and testing examples and also more classes of objects. Also, we would like to test our approach with manually specified keypoints for the prototype images where the user would be able to specify which parts of the object are useful for classification. Lastly, we will try to incorporate spatial constraints between keypoints for the matching algorithm, which should make the classification more robust.

9 Conclusion

In this article, we described and evaluated a method for keypoint selection, which is based on the attention module from [4]. We evaluated the method in proof-of-concept experiments and showed that it could select a small subset of relevant keypoints from a large set of generic keypoints. We also showed the importance of the right distance function when matching individual keypoints. We hope that such developments will enable the use of keypoints in tasks where they are not standardly used.

References

[1] G. Lowe, "Sift-the scale invariant feature transform," *Int. J*, vol. 2, pp. 91–110, 2004.

[2] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006.

[3] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[4] Y. Tang, D. Nguyen, and D. Ha, "Neuroevolution of self-interpretable agents," *arXiv preprint arXiv:2003.08165*, 2020.

[5] M. Cho, K. Alahari, and J. Ponce, "Learning graphs to match," in *Proceedings of the IEEE Interational Conference on Computer Vision*, 2013.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[7] X. Liu and M. Milanova, "Visual attention in deep learning: a review," *Int. Robot. Automat. J*, vol. 4, pp. 154–155, 2018.

[8] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[9] V. Mnih, N. Heess, A. Graves, *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, pp. 2204–2212, 2014.

[10] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.

[11] G. Elsayed, S. Kornblith, and Q. V. Le, "Saccader: improving accuracy of hard attention models for vision," in *Advances in Neural Information Processing Systems*, pp. 702–714, 2019.

[12] W.-C. Lin, C.-F. Tsai, Z.-Y. Chen, and S.-W. Ke, "Keypoint selection for efficient bag-of-words feature generation and effective image classification," *Information Sciences*, vol. 329, pp. 33–51, 2016.

[13] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih, "Unsupervised learning of object keypoints for perception and control," in *Advances in neural information processing systems*, pp. 10724–10734, 2019.

[14] S. Suwajanakorn, N. Snavely, J. J. Tompson, and M. Norouzi, "Discovery of latent 3d keypoints via end-to-end geometric reasoning," in *Advances in neural information processing systems*, pp. 2059–2070, 2018.