Transformation-Invariant Convolutional Jungles

Dmitry Laptev
ETH Zurich, Switzerland
dlaptev@inf.ethz.ch

Joachim M.Buhmann
ETH Zurich, Switzerland
jbuhmann@inf.ethz.ch

Abstract

Many Computer Vision problems arise from information processing of data sources with nuisance variances like scale, orientation, contrast, perspective foreshortening or – in medical imaging – staining and local warping. In most cases these variances can be stated a priori and can be used to improve the generalization of recognition algorithms. We propose a novel supervised feature learning approach, which efficiently extracts information from these constraints to produce interpretable, transformation-invariant features. The proposed method can incorporate a large class of transformations, e.g., shifts, rotations, change of scale, morphological operations, non-linear distortions, photometric transformations, etc. These features boost the discrimination power of a novel image classification and segmentation method, which we call Transformation-Invariant Convolutional Jungles (TICJ). We test the algorithm on two benchmarks in face recognition and medical imaging, where it achieves state of the art results, while being computationally significantly more efficient than Deep Neural Networks.

1. Introduction

Human visual perception proves to be extremely stable to a broad class of variations in scenes. If objects in images are rotated, or scaled, or even non-linearly distorted – in most cases we still can recognise these objects. If we are in advance aware of transformations that can occur in a dataset, then this information can be used as prior to design a better recognition algorithm with higher generalization capacity and therefore, higher accuracy than agnostic learners. Being capable of generalizing over different transformations is a very important property of any Machine Learning approach, and especially of Computer Vision algorithms.

The set of transformations to be considered highly depends on the task that one has to solve. Some common examples are presented in figure 1, but a possible transformation set is neither limited to these examples, nor needs to include all of them. For example, rotation-invariance should be used wisely for digit recognition task, since rotating the digit "6" by 180° could lead to its confusion with "9". However, smaller rotations of up to $\pm 15°$ proved to significantly improve accuracy in the MNIST classification benchmark [6]. Scale-invariance can also harm classification performance if object size is at least somehow informative, for example, in case of classifying healthy cells from cancer cells [24].

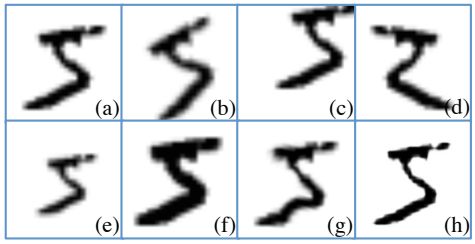

Figure 1. Example of transformations ϕ that are usually considered in Computer Vision tasks applied to a handwritten digit "5" from MNIST dataset [16]. (a) shows the original image X, (b)–(h) show different transformation results $\phi(X)$: rotation (b), shift (translation) (c), reflection (d), scaling (e), morphological operations (f), non-linear distortions (g), brightness, contrast change (h).

Once a set of transformations for a given task is known and fixed, there are three main ways to incorporate this prior knowledge: (i) change the dataset itself, (ii) use transformation-invariant features or (iii) modify the learning algorithm. Arguably the most popular approach is the first one, which enlarges the original dataset by adding the images that were transformed according to our prior beliefs. This strategy allows the recognition algorithm to observe all the instantiations of the data and, in case of flexible models to adapt to all considered transformations.

Enlarging the dataset, however, implies also to significantly extend the training time and it requires extensive computational resources. But very large data sets pose only one of the problems: to cope with larger variations in datasets, data analysts usually have to increase the number of parameters in modeling, requiring even more training time, more memory, and posing the risk of over- or under-

fitting. Therefore, current research investigates advanced techniques to avoid this pitfall, as reviewed in section 2.

The proposed method is inspired by the idea of the pooling operation, that preserves some local invariances and seems to be biologically plausible [22]. We follow this idea to incorporate prior knowledge about the transformations through learning transformation-invariant features of the images. We define these features through the convolutional kernels, but instead of convolving the image itself with the kernel, we propose to compute the maximum over many convolutions: with the given image, and with all the considered transformations of this image. This nonlinear operation assures transformation-invariance, as the value of the maximum is exactly the same for the original image, and the image that was transformed (see lemma 1 for more details).

Section 3.2 discusses how these features can be efficiently trained in a supervised manner to fit the needs of the specific task, and how they can be regularized to get interpretable transformation-invariant features that generalize well.

The proposed algorithm TICJ uses this transformation-invariant feature learning procedure to build an image classification algorithm (or per-pixel segmentation algorithm). This is done by iteratively learning the features and combining them together in a feedforward modification of the Decision Jungles algorithm [26]. Comparing to Decision Trees [21], this algorithm proves to better prevent overfitting, and also efficiently works with limited memory constraints. The combination of the proposed feature learning algorithm and the proposed modification of Decision Jungles allows us to achieve state of the art results with modest training time, as described in section 3.3 in detail.

The main properties and contributions of the proposed method are the following:

- Transformation-invariant feature learning allows us to incorporate any types of transformation invariances as prior constraints. This method results in good generalization without enlarging the original dataset size or the parameter space.

- Regularization is enforced in two different ways by the learning method and it serves the purpose of producing interpretable features. These features are easy to debug, since they are defined through convolutional kernels and they support visual inspection (see figure 2). Regularization also helps when the dataset is small: as we show in the experimental section 4.1, TICJ can efficiently be applied for datasets starting from only tens of images.

- The final classification algorithm is computationally very efficient and, unlike many state of the art techniques, can be easily run on a single CPU within a modest training time. In our experiments, the proposed method is up to two orders of magnitude faster than, for example, Deep Neural Networks, while achieving state of the art classification performance (see section 4.2).

- The method has only few hyperparameters, and we show in section 3.4 how they can be efficiently tuned. This simplicity of the method makes it highly suitable for plug-and-play experiments in comparison to many other modern Computer Vision techniques.

We also propose a modification of the node clustering technique for Decision Jungles and thereby, we overcome the problem of global clustering that produced poor results as mentioned in the original Decision Jungles paper [26]. This contribution is discussed in more details in section 3.3.

2. Related work

2.1. Predefined features

One of the easiest ways to incorporate partial transformation-invariances is to use special types of predefined, often “hand-crafted” features, i.e., the scale-invariant feature transform (SIFT) [17] or its rotation-invariant modification RIFT (rotation-invariant feature transform) [14] have proved to boost performance in a broad range of image processing applications and imaging modalities.

These features are designed to be general-purpose and transformation-invariant, and they satisfactorily solve the task in many cases. However, they are limited in two ways: they only can incorporate very specific transformations, and they do not adapt to the task being solved.

One of the ways to overcome the second deficit is to design hand-crafted features for the specific task. Line filter transform [23] for blood vessel segmentation is one of the examples of domain-specific features, which is also rotation-invariant. But designing the features manually is time-consuming and expensive while not always possible.

The proposed method, in contrast to predefined features, not only learns the features in a supervised manner, but also allows one to incorporate any types of invariances.

2.2. Feature learning

Instead of designing features for every different task, one could learn these features automatically such that for every dataset a set of learned features would be the most representative and/or discriminative. Four approaches that follow this idea are ”bag of visual words” (BOVW) [20], Convolutional Decision Trees (CDT) [12], Sparse Coding [28] and different types of Neural Networks.

BOVW does not distinguish the positions in which the ”visual word” occurs, and therefore it is a shift-invariant

method. Rotation-invariance can be achieved [29] with just few modifications.

CDT trains the features as convolutional kernels, which resembles our proposal. In contrast to CDT, we do not directly optimize the information gain of the split defined by the features, but solve a convex approximation to it formulated as a regularized least squares problem. This optimization produces comparable results to the information gain maximization procedure, but allows us to find the solution significantly faster. Furthermore, CDT does not allow to incorporate transformation-invariance.

Sparse Coding algorithm can be also modified to learn overcomplete shift-invariant image representation as presented in the paper [28]. We consider neural networks in the following section.

The proposed algorithm incorporates various ideas of the above methods to learn features from data and, simultaneously, to adapt to the specific task. But unlike both BOVW and Sparse Coding algorithms, it allows us to incorporate many different types of transformations.

2.3. Transformation-invariant neural networks

Deep Neural Network architectures are the richest model classes used nowadays in Computer Vision and they enable very impressive results in many tasks. Because of their richness, many modifications can be implemented to incorporate different types of prior knowledge in the training process itself. Not surprisingly, transformation-invariance is actively discussed also in the field of "Deep Neural Networks".

The most commonly used property of Convolutional Neural Networks that enables some transformation invariance is a subsampling layer [15] with *max-pooling*. Because the maximum is taken over the neighbouring pixels, local one-pixel shifts usually do not change the output of the subsampling layer. A more general pooling operation [22] allows one to also consider local rotation and scale changes. As usually many layers are stacked in a hierarchy on top of each other, the window size for local invariances increases.

Other techniques that support invariances to a rich transformation class include topographic filter maps [11] (learns features invariant to rotation, shift and scale changes) and the algorithm presented in [27] (local transformations that can be approximated as linear transformations). However, neither of these approaches can learn arbitrary set of transformations. This challenge has been already discussed in section 1. Another simple approach that works without enlarging the dataset and the model size is presented in [7] and [6]. The idea is to train different models with the same topology but using different datasets: the original dataset, and the transformed datasets (one model for every transformation considered). Then either the weights of these models are averaged to produce one new model, or the outputs of the networks vote for a majority, forming an ensemble of models. This last approach is widely used and we compare our algorithms to it as a baseline in one of our experiments (see section 4.2).

3. Method description

3.1. Notation

Let us consider an image classification dataset with K classes that consists of N images. Let $X_i \in \mathbb{R}^{w \times w}$ be the i-th image in this dataset represented by a square real-valued matrix of pixel intensities, where $i = 1, \ldots, N$ and w is the size of the image. For simplicity we consider square images, however, the method naturally generalizes also to rectangular images.

Every image X_i has an assigned class $y_i \in \{1, \ldots, K\}$. The task of an image classification algorithm is to return a class estimate $\hat{y}$ for a new unobserved image X.

To address segmentation tasks, we employ the commonly used patch classification strategy: consider X_i to be a patch around pixel i, and y_i to be the corresponding pixel class (segment index). Then, the algorithm should return the class estimate for every pixel based on the appearance of the surrounding $w \times w$ pixel area. The patch size w is an application dependent parameter that we select by cross-validation.

In homogeneous coordinates, the image X_i is described by a vector $x_i \in \mathbb{R}^{w^2+1}$ with $x_{i,1} \equiv 1$.

Assume Φ to be a set of all considered transformations. $\Phi = \{\phi_1, \ldots, \phi_T\}$, where ϕ_t denotes a transformation function and T specifies the number of transformations. If X is an image of the dataset, then $\phi(X)$ represents a transformed image of the same size $w \times w$. For simplicity of notation, $\phi(x)$ also denotes an extended vectorized representation of the transformed image $\phi(X)$. Φ always includes the identity transformation $\phi_0 : \phi_0(X) \equiv X$.

The reader should notice that ϕ can represent either one of the simple transformations shown in figure 1, or the combination of these transformations. I.e. ϕ_3 could be the composition of ϕ_1 and ϕ_2: $\phi_3 = \phi_1 \circ \phi_2$ means that $\phi_3(\cdot) = \phi_1(\phi_2(\cdot))$. Because of a discrete image representation in computers, the set of possible transformations can always be considered finite, even though it can be very large.

3.2. Transformation-invariant feature learning

3.2.1 Feature definition and properties

As discussed in section 1, we parametrize a feature with a convolutional kernel $\theta \in \mathbb{R}^{w^2+1}$. The value of the feature for an image X is given by:

$$f_\theta(x) = \max_{\phi \in \Phi} \theta^T \phi(x) \tag{1}$$

Because of the maximum operation, this equation in most cases gives exactly the same result $f_\theta(x)$ for the image X itself, and for the transformations of this image $\phi(X)$. Lemma 1 formulates the conditions on the set Φ for which this holds true.

Lemma 1. *The feature of the image X defined in equation 1 is transformation-invariant if the set Φ of all possible transformations forms a group, i.e. satisfies the axioms of closure, associativity, invertibility and identity.*

Proof. In order to prove this statement, the value of the feature has to be the same for all the transformations of the image. Since Φ always contains an identity transformation, we can compare the value of the feature with the value of the feature for the identity transformation ϕ_0. So we need to show that $f_\theta(\psi(X)) = f_\theta(\phi_0(X)) = f_\theta(X), \forall \psi \in \Phi$.

For any transformation $\psi \in \Phi$ the following holds:

$$f_\theta(\psi(x)) = \max_{\phi \in \Phi} \theta^T \phi(\psi(x)) = \max_{\varphi = \phi \circ \psi:\ \phi \in \Phi} \theta^T \varphi(x).$$

The closure axiom implies

$$\{\phi \circ \psi : \phi \in \Phi\} \subseteq \Phi. \tag{2}$$

On the other hand, invertibility axiom assure the existence of an inverse, $\forall \psi \in \Phi, \exists \psi^{-1}$. Furthermore, $\Phi \supseteq \{\phi \circ \psi^{-1}, \phi \in \Phi\} =: \Psi$ (as for Ψ we select only the elements of the set Φ that can be represented through a composition with ψ^{-1}).

Therefore,

$$\{\phi \circ \psi : \phi \in \Phi\} \supseteq \{\phi \circ \psi : \phi \in \Psi\} =$$
$$\{\phi \circ \psi^{-1} \circ \psi : \phi \in \Phi\} = \{\phi : \phi \in \Phi\} = \Phi. \tag{3}$$

Equations 2 and 3 show that $\{\phi \circ \psi : \phi \in \Phi\} \equiv \Phi$ and therefore, the set over which the maximum is taken stays the same, which shows that $f_\theta(\phi(X)) \equiv f_\theta(X)$. □

The statement of the lemma is satisfied for many Computer Vision tasks: basically all the simple transformations shown in figure 1 as well as their compositions form a group. The most common examples of the transformation sets that do not satisfy this property include local shifts (jittering) and local rotations. For example, if one wants to consider only one pixel shifts, then the closure axiom of the group does not hold: one pixel shift applied twice gives two-pixel shift, which is not in a transformation set.

However, one can easily modify the definition of the feature such that it stays transformation-invariant with respect to local transformations:

$$f_\theta(x) = \max_{\phi, \psi \in \Phi} \theta^T \phi(\psi(x)) \tag{4}$$

This formulation allows us to relax the closure axiom of the whole set to the closure of only two elements of the set. Features defined by equation 4 are invariant to every transformation in Φ (but not in $\{\phi \circ \psi : \phi, \psi \in \Phi\}$) if the transformations set Φ contains all the inverse elements and the identity element. The proof of the last statement stays almost the same, but with a different set over which the maximum is taken.

Therefore, if one wants to consider only local transformations and lets Φ to contain, for example, one pixel shift to the left and to the right (together with an identity transformation), then the set over which the maximum should be taken includes shifts by one *or two* pixels.

In the following, we consider the definition 1 of a feature to simplify our notation.

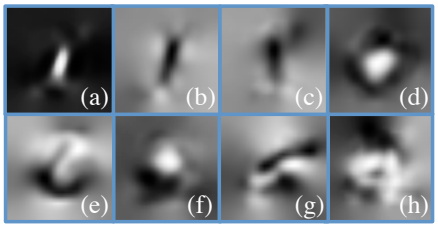

Figure 2. Examples of different kernels θ learned with TICJ algorithm applied to a neuronal segmentation dataset. One could see that the features are relatively meaningful: features (a)–(c) detect direct lines (this correspond to straight membranes in the dataset), (d) denotes the contrast of the center pixels comparing to the surroundings, (e) and (f) detect corners and curvatures (non-straight membranes), (g) and (h) – membrane conjunctions and textures of neuronal tissue (high-frequency features).

3.2.2 Feature learning

Lemma 1 shows that the features formulated in equation 1 are transformation-invariant. However, one also needs to establish the procedure of learning the parameters θ of the feature.

Assume that we select two classes $c_1, c_2 \in [1, \dots, K]$ and we want to separate the images of these classes. We propose to find the parameter vector θ by solving the following optimization problem:

$$\theta = \arg\min_\theta E(\theta) = \arg\min_\theta \ \lambda ||\Gamma\theta||_2^2 + \sum_{i:\ y_i = c_1 \text{ or } y_i = c_2} \left(f_\theta(X_i) + [y_i = c_1] - [y_i = c_2]\right)^2 \tag{5}$$

Here $f_\theta(x)$ is a feature defined in 1 and $[\cdot]$ refers to Iverson brackets, that are equal to 1 if $\cdot$ is true and zero otherwise. Following this notation, $[y_i = c_1] - [y_i = c_2]$ is equal to 1 if $y_i = c_1$ and equal to -1 if $y_i = c_2$. Matrix $\Gamma \in \mathbb{R}^{2w(w-1) \times (w^2+1)}$ is a matrix of a 2D differentiation operator in a vectorized space, that is a Tikhonov regularization matrix. Penalizing the gradient of the kernel enforces

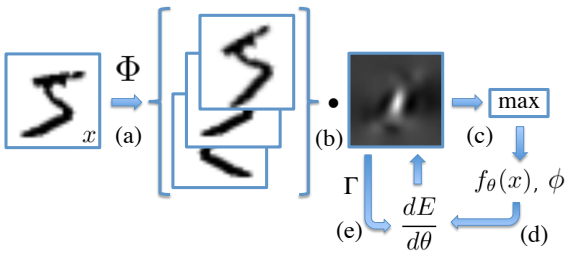

Figure 3. Partial visualization of the feature learning process. For all the images x, we compute the transformations $\phi(x), \forall \phi \in \Phi$ (a). Every image after transformation is convolved linearly with the current kernel vector θ (b). That gives the response for every transformation, from which we select then the maximum $f_\theta(x)$ and the corresponding transformation ϕ that gives maximum response (c). These values are then used to compute the functional value and the gradient of the functional value (d) when combined with the regularization term (e). The gradient step $\frac{dE}{d\theta}$ then updates the value of the feature parameters θ.

the kernel to be smooth. λ is a regularization parameter that controls the trade-off between the goodness of separation and the smoothness of the kernel learned.

Regularization serves two main goals. First of all, it ensures interpretability of the inferred kernels (see figure 2). Second, from an optimization point of view, a strictly concave and differentiable regularization term increases the convergence speed of the gradient descent optimization algorithm.

In order to efficiently find the minimum of $E(\theta)$, we also compute the subgradient of the functional 5:

$$\frac{dE}{d\theta} = \sum_{\substack{i:\ y_i=c_1 \\ \text{or } y_i=c_2}} 2\left(f_\theta(X_i) + [y_i = c_1] - [y_i = c_2]\right)\phi_i(X_i) + 2\lambda\Gamma^T\Gamma\theta \tag{6}$$

where $\phi_i = \arg\max_{\phi\in\Phi} \theta^T\phi(X_i)$ – is a transformation that gives the maximum response for an input image X_i.

Based on the formulas 5 and 6 one can implement an optimization algorithm that finds the optimum value of θ for a given dataset $\{X_i, y_i\}$ and for two selected classes c_1 and c_2. It is important to notice that the problem 5 is not continuously differentiable because of the maximum in the definition of $f_\theta(X)$, however, it is convex and therefore one is guaranteed to find the global optimum of the problem. In our experiments we explored different optimization techniques, but finally selected the L-BFGS optimization subroutine [19] as it always yielded the highest convergence speed for the tasks we consider. The constructive version of the algorithm is sketched in figure 3.

One important question is how one selects c_1 and c_2. We propose to take them at random with the probabilities p_c proportional to the presence of the class c in the dataset: $p_c \sim |\{i : y_i = c\}|$. This choice assures that we try to separate the largest classes with high probability, but also leaves room for randomization of the algorithm, which is important for ensemble learning discussed in section 3.3. There, we also discuss the problem of selecting the regularization parameter λ.

3.3. TICT and TICJ: Transformation-Invariant Convolutional Trees and Jungles

Section 3.2 shows how to learn the parameters of a transformation-invariant feature that splits the dataset into two subsets: one subset consists of the images $X_i : f_\theta(X_i) > 0$, another of images $X_i : f_\theta(X_i) \leq 0$. That means that the feature defines a split predicate on the space of images, and therefore can be used in algorithms such as decision trees.

This section discusses how to learn these features and combine them in an iterative feedforward manner to build a final image classification algorithm.

3.3.1 Transformation-Invariant Convolutional Trees

Following the idea of decision trees [21], we learn one feature and then split the whole dataset into two subsets according to the predicate defined by this feature. To each of the subsets the same idea can be applied recursively until a termination criterion is satisfied. We call this algorithm Transformation-Invariant Convolutional Trees (TICT).

Formally, we start with a root node that accepts the whole dataset $\{X_i, y_i : i = 1, \dots, N\}$ as input, and trains θ to define a root feature. Then we split the training dataset into subsets l_1 and l_2: $l_1 = \{i : f_\theta(X_i) > 0\}$, $l_2 = \{i : f_\theta(X_i) \leq 0\}$. For this first layer we define a set of leaves $L = \{l, r\}$.

Then each new layer is built recursively as follows.

- For every leaf $l \in L$ we train new feature parameters θ, but using only a subset of the original dataset defined by indices in l.
- Then we split l to $l_1 = \{i : i \in l, f_\theta(X_i) > 0\}$ and $l_2 = \{i : i \in l, f_\theta(X_i) \leq 0\}$.
- If $|l_1| > 0$ and $|l_2| > 0$ (the split is non-trivial), then the new leaves set is defined as $L \cup \{l_{j,1}, l_{j,2}\} \setminus l_j$, otherwise it does not change.
- If $|l_1| = 0$ or $|l_2| = 0$, but $|\{c : \exists i \in l \text{ s.t. } y_i = c\}| > 1$ (this denotes that the leaf l contains objects of at least two different classes), that means that the features are not flexible enough to separate the dataset and we increase its flexibility by decreasing the value of λ (for example, multiply it by $\frac{2}{3}$).

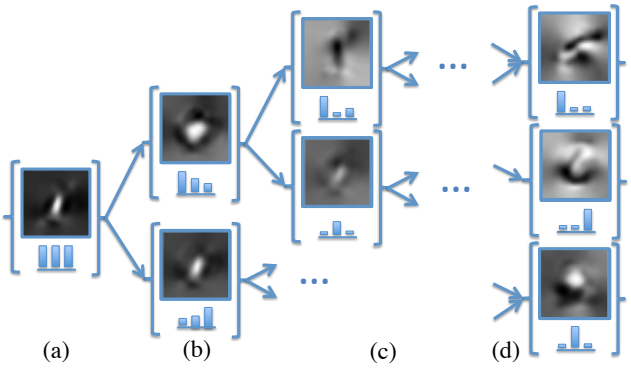

Figure 4. A visualization of TICJ training process. Each node is represented with feature parameters θ and a histogram h of input object classes (for simplicity we consider three classes here). (a) shows the root node, for which the whole dataset is an input. Using the learned feature f_θ – the dataset is split in two subsets to serve as input for two other nodes (b). The algorithm proceeds by splitting the dataset until the maximum number width M is achieved (c). Then some of the data subsets can be joined together with a histogram clustering technique (d).

We add layers with the above procedure until the maximum number of iterations is reached. If at some iteration step $|\{c : \exists i \in l \text{ s.t. } y_i = c\}| = 1$ for every $l \in L$ – the training dataset is perfectly separated and the algorithm terminates.

The classification of new image X with TICT is achieved in exactly the same way as with decision trees: we go from the root node following the splits to the leaves, and then return the majority class of the objects in this leaf. The proposed algorithm is similar to oblique decision trees [18]. However, unlike oblique decision trees, the features in our case do not form a linear combination of all the pixel intensities, and therefore TICT does not belong to this category. We use TICT for comparison, but the final algorithm uses a modification of it inspired by Decision Jungles [26].

3.3.2 Transformation-Invariant Convolutional Jungles

There are two main problems with the previous approach:

- first, it easily overfits the data if the number of splits is large and the sizes of the leaves are small;
- second, the size of TICT grows very fast, causing major efficiency and memory issues (if we add 20 layers, in worst case we need to train 2^{21} features).

In order to overcome both of these issues, we propose TICJ algorithm based on the modification of the Decision Jungles algorithm. The idea of TICJ is very simple: after adding one layer, we perform the clustering of leaves in L and join similar leaves together where the similarity of leaves is measured as the similarity of the histograms of the classes present in a leaf. We merge leaves only if the leaves set size $|L|$ is greater than some constant M. We also generate a new layer in a feedforward manner, so after joining the leafs, we do not retrain the features. That allows us to spend up to two times less training time, and produces very similar results to a two-step procedure in our experiments. The scheme of the algorithm is sketched in figure 4.

The second extension of the original decision jungles paper is how we perform clustering. The paper [26] suggests two clustering technique: a global and a randomized greedy, and claims that a global clustering technique performs worse. We experienced very similar behaviour and discovered a possible reason behind that: very often global clustering joins the leafs that were separated just before with the feature learned. For example, quite a common case is that θ learned in one layer splits l into l_1 and l_2, and then the clustering algorithm groups l_1 and l_2 again together. That means that in the next layer θ will be trained again with the same data, and with high probability will yield the same results, getting the algorithm stuck in this loop.

To overcome this issue, we propose to forbid the clusters consisting of two leaves that were just split. That can be easily implemented by just setting the distance of these leaves to be infinite before executing the clustering algorithm. Formally we define the distance between leafs $D(l_j, l_i)$ as either $+\infty$ if l_j and l_i originate from one set l_k, or $D(l_j, l_i) = \frac{1}{2}\left(D_{KL}(h_j||h_i) + D_{KL}(h_i||h_j)\right)$ otherwise. Here $D_{KL}(\cdot||\cdot)$ stands for Kullback-Leibler divergence [10], and $h_j = [|\{i \in l_j : y_i = 1\}|, \ldots, |\{i \in l_j : y_i = K\}|]$ – a histogram representation of the leaf classes present in a leaf l_j. Then we apply agglomerative clustering [8] with the defined metric to get clusters $1, \ldots, M$ and redefine the leaf set to be $L = \{\cup_{l \in \text{ cluster } 1} l, \ldots, \cup_{l \in \text{ cluster } M} l\}$.

One can also construct an ensemble of trained jungles. Such an ensemble may slightly increases the recognition performance in some cases and reduces the variance of the resulting classifier. This improvement is caused by the randomness involved into constructing every instance of TICJ: classes for separation are taken at random. One can also use only a subset of objects or a subset of transformations for TICJ training to further diversify the trained models and benefit more from averaging their outputs.

3.4. Algorithm parameters discussions

In this section we demonstrate how to better understand the parameters and how to set them wisely without harming the performance and the efficiency of the algorithm. There are three main parameters in the proposed algorithm: (i) a set of transformations Φ, (ii) the regularization parameter λ considered during feature learning, and (iii) the maximum TICJ width M.

The maximum number of iterations is also a hyperparameter, but it is less important as it does not need to be specified in advance: if the performance on the validation dataset is still improving, one can always add more layers.

Φ – a set of transformations – depends on the task being solved and almost always can be selected in advance, as discussed in section 1. We also want to note that Φ partly serves the regularization purpose. When the size of the set Φ increases, then the learned feature f_θ is expected to have less degrees of freedom. Therefore one should be careful when selecting a large set Φ as it can prevent flexible features from being learned.

This model selection choice is, however, not an issue when we determine the regularization parameter λ. One can start with a large value of λ to learn very smooth kernels corresponding to low-frequency features. As we discuss in section 3.3, if learning θ gives only trivial splits, we decrease the value of λ (usually just multiply it by $\frac{2}{3}$) and start to discover also high-frequency features. That gradually increases the complexity of the features as we go down the layers hierarchy.

The maximum width of the jungle M is the only remaining parameter that defines the topology of TICJ. This parameter also significantly contributes to the control of the bias-variance trade-off: if it is small, it prevents overfitting, but the algorithm usually appears to be less flexible. If it is chosen too large then TICJ adapts to fluctuations. We propose to use the following heuristic:

- start with a small value of M (we usually take $M = 3K$, where K is the number of classes),
- train the algorithm by adding more layers and observe the validation error;
- if the validation error stops decreasing, enlarge M without retraining the whole TICJ and continue adding layers (just the new layers would be wider).

This process can be repeated until the algorithm starts overfitting, which is usually indicated by the increase in a validation set error.

4. Experiments

In this section we present the experimental results on two publicly available computer vision datasets: (i) the Yale face recognition dataset [2] and (ii) the Neuronal structures segmentation dataset [4]. Both datasets include large intra class variabilities (see examples in figure 5), but also contain some transformation-invariances, which we exploit with the proposed algorithm.

In face recognition benchmark we achieve slightly better results, than the state of the art algorithms: 0.3% increase in accuracy (here we consider only the methods that use no additional training data).

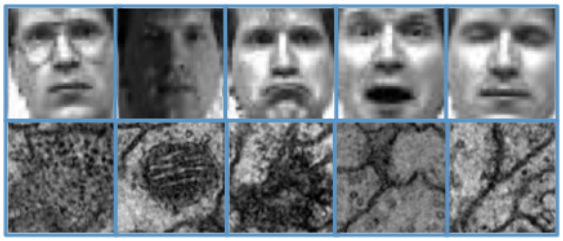

Figure 5. Example cropped images from Yale face recognition dataset (top row) and example patches from Neuronal structures segmentation dataset (bottom row). The Yale dataset includes large variations in pose, facial expression, illumination and sometimes includes obstacles (glasses). Patches of neuronal tissue sometimes clearly indicate membranes (the last two images), but in many cases the images display very unclear and blurred structures that are hard to detect even for a trained human expert.

Figure 6. An example implementation of a transformation that changes illumination. The original image is per-pixel divided by a blurred version of itself. This makes some originally dark regions a little lighter.

In structure segmentation benchmark, we exactly match the performance of the current state of the art algorithm, which is Convolutional Neural Networks [5], but obtain the results orders of magnitudes faster.

4.1. Face recognition

Original Yale face recognition dataset contains 165 grayscale images of 15 individuals (and therefore has $K = 15$ classes). There are 11 images per subject, one per different facial expression (normal, happy, sad, sleepy, surprised, and wink) or configuration (left-light, center-light, right-light, with/without glasses).

We follow the most commonly adopted experimental setup [3, 9, 25] and average the results over 50 random splits into training and test sets. Splits are performed independently for all images of a particular person. For training we select five images per person, and use the other six for testing. We also use the cropped version of the dataset [3] since most publications with competing methods follow this protocol.

We run both TICT and TICJ with the set Φ of transformations that includes small shifts (up to two pixels in each side) and illumination changes. We implement illumination changes simply through dividing the original image by the very blurred one (see figure 6 for example). For blurring we use Gaussian kernels with width 8 and 16. Other parameters

are selected as described in section 3.4.

As baselines we select two state of the art methods that achieve the best results for this experimental setup: spatially smooth subspace learning [3] and orthogonal rank one tensor projections [9]. The results are presented in table below. The table also includes neural networks on pretrained features [25] that performs better than the proposed TICJ. However, this method uses additional data for the feature learning process, and therefore a comparison is questionable. Apart from this method, TICJ achieves better results than state of the art methods using no additional data. TICT overfits the data and performs significantly worse.

Method	Error (%)
Cai et al. [3]	18.3
Cai et al. [3] (updated)	14.7
Hua et al. [9]	13.2
Shan et al. [25]	8.2
TICT (ours)	18.4
TICJ ($\Phi = \varnothing$) (ours)	16
TICJ (ours)	12.9

4.2. Neuronal structure segmentation

The neuronal membrane segmentation dataset was used for ISBI 2012 challenge [1]. It consists of 30 training and 30 test images of neuronal tissue captured with serial section transmission electron microscopy. The task is to perform pixel segmentation into two classes: cell membranes and inner parts of the neuron. Because the classes are imbalanced, the accuracy is measured in F-score: a commonly used metric that combines precision and recall.

From the neurological experts we know, that membrane appearance does not depend on the orientation of the membrane, and therefore we can safely include 360° rotations in the set of transformations Φ. We sample rotations every 15 degrees, resulting in 24 transformations considered.

We perform training on 50000 pixel patches X selected at random from the training images together with the labels of the corresponding pixel y. We select the patch size to be 31×31 pixels ($w = 31$).

As baselines, we select the methods that won the first and the second place in the challenge [1]. The first method is an ensemble of convolutional neural networks (CNN) [5]. The second method is a random forest per-pixel classifier with a huge number of features and cross-image priors (RF) [13]. We also include recently proposed convolutional decision trees (CDT) [12] as they outperform RF.

Method	1–F-score (%)	Training time
RF [13]	7.9	unknown
CDT [12]	6.8	8 hours (CPU)
CNN [5]	6.0	7 days (GPU)
TICT (ours)	6.7	2.5 hours (CPU)
TICJ (ours)	6.0	3 hours (CPU)

The results of the experiment are presented in table above: TICJ matches the state of the art results of Convolutional Neural Networks and consistently outperforms other methods. It is also very important to notice the highly significant speedup during training, where the training time is orders of magnitude smaller for TICJ then for CNN. CNN are reported to train for about one week on a GPU cluster, and the estimated time is one year on a single CPU. TICJ, on the contrary, can be trained in one CPU within three hours.

5. Conclusions

Invariance to different types of transformations is required in many domains of Machine Learning and Computer Vision. The prior knowledge about nuisance transformations that are reflected in visual data sets can be incorporated into a learning process to achieve better accuracy and higher generalization capacity.

In this paper we propose a novel image classification algorithm called Transformation-Invariant Convolutional Jungles (TICJ). The method is based on transformation-invariant features, inspired by a pooling operation similar to HMAX, but parametrized with a global convolutional kernel. To assure that these features are transformation-invariant, we take the maximum response value over the transformations considered (see lemma 1).

We learn the parameters of these features by solving a convex optimization problem, that incorporates the term enforcing the separation of the dataset, and the term penalizing the complexity of the learned kernel. These features are then used as predicates to form the TICJ algorithm.

Regularization is enforced in TICJ through transformation-invariance constraints, gradient regularization and through the limitation on the maximum width of the final TICJ. These design constraints render the learned features interpretable and ensure satisfactory generalization even for small datasets.

We test the proposed approach on two very different datasets: (i) Yale face recognition dataset, that is very small (15 classes, 5 images per class for training), and (ii) Neuronal structures segmentation dataset (contains tens of thousands of samples for each of two classes). In both datasets we achieve state of the art results. On the Yale dataset we outperform the competitors by at least 0.3%, if we consider the algorithms that do not use additional training data (the only algorithm with superior performance uses features pretrained on another dataset). For the Neuronal membrane segmentation dataset we achieve the same F-score as Convolutional Neural Networks approach, but we train TICJ within 3 hours in a single CPU, comparing to about one week CNN training on a GPU cluster. The paper demonstrates that incorporating transformation-invariances in the feature learning process can significantly boost the performance while being computationally very efficient.

References

[1] Isbi challenge: Segmentation of neuronal structures in em stacks (http://brainiac2.mit.edu/isbi_challenge/). 8

[2] P. N. Belhumeur, J. P. Hespanha, and D. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997. 7

[3] D. Cai, X. He, Y. Hu, J. Han, and T. Huang. Learning a spatially smooth subspace for face recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007. 7, 8

[4] A. Cardona, S. Saalfeld, S. Preibisch, B. Schmid, A. Cheng, J. Pulokas, P. Tomancak, and V. Hartenstein. An integrated micro-and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLoS biology*, 8(10):e1000502, 2010. 7

[5] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012. 7, 8

[6] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012. 1, 3

[7] C. L. Giles and T. Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied optics*, 26(23):4972–4978, 1987. 3

[8] J. Goldberger, S. Gordon, and H. Greenspan. Unsupervised image-set clustering using an information theoretic framework. *Image Processing, IEEE Transactions on*, 15(2):449–458, 2006. 6

[9] G. Hua, P. A. Viola, and S. M. Drucker. Face recognition using discriminatively trained orthogonal rank one tensor projections. In *Computer Vision and Pattern Recognition CVPR 2007. IEEE Conference on*, pages 1–8. IEEE, 2007. 7, 8

[10] T. Kailath. The divergence and bhattacharyya distance measures in signal selection. *Communication Technology, IEEE Transactions on*, 15(1):52–60, 1967. 6

[11] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition CVPR 2009. IEEE Conference on*, pages 1605–1612. IEEE, 2009. 3

[12] D. Laptev and J. M. Buhmann. Convolutional decision trees for feature learning and segmentation. In *Pattern Recognition*, pages 95–106. Springer, 2014. 2, 8

[13] D. Laptev, A. Vezhnevets, S. Dwivedi, and J. M. Buhmann. Anisotropic sstem image segmentation using dense correspondence across sections. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2012*, pages 323–330. Springer, 2012. 8

[14] S. Lazebnik, C. Schmid, J. Ponce, et al. Semi-local affine parts for object recognition. In *British Machine Vision Conference (BMVC'04)*, pages 779–788, 2004. 2

[15] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995. 3

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[17] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 2

[18] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994. 6

[19] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980. 5

[20] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 2

[21] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 2, 5

[22] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999. 2, 3

[23] K. Sandberg and M. Brega. Segmentation of thin structures in electron micrographs using orientation fields. *Journal of structural biology*, 157(2):403–415, 2007. 2

[24] P. J. Schüffler, T. J. Fuchs, C. S. Ong, V. Roth, and J. M. Buhmann. Computational tma analysis and cell nucleus classification of renal cell carcinoma. In *Pattern Recognition*, pages 202–211. Springer, 2010. 1

[25] H. Shan and G. W. Cottrell. Looking around the backyard helps to recognize faces and digits. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 7, 8

[26] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. Winn, and A. Criminisi. Decision jungles: Compact and rich models for classification. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 234–242. Curran Associates, Inc., 2013. 2, 6

[27] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1311–1318, 2012. 3

[28] J. Yang, K. Yu, and T. Huang. Supervised translation-invariant sparse coding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3517–3524. IEEE, 2010. 2, 3

[29] Y. Yang and S. Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 270–279. ACM, 2010. 3