Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehr- und Forschungsgebiet Informatik VIII
Computer Vision
Prof. Dr. Bastian Leibe

Seminar Report

Decision Jungles

Tobias Pohlen
Matriculation Number: 308743

July 2014

Advisor: Alexander Hermans

Abstract

In this paper, we present and discuss decision jungles as proposed by Shotton et al. at NIPS 2013. Decision jungles are ensembles of randomly trained decision DAGs (directed acyclic graphs). The concept is closely related to that of random forests as proposed by Breiman. For this reason, while discussing the topic, we compare it random forests in order to highlight the similarities as well as the differences. The main contribution of this paper is the derivation of an efficient implementation of the LSEARCH optimization algorithm. Using our resulting open source library, we perform several experiments in order to (1) investigate the influence different parameter settings have on the resulting classifier; (2) compare the performance of decision jungles and random forests for the task of multi class classification on a variety of data sets.

Contents

1 Motivation

Machine learning techniques and algorithms have become a major building block of modern computer systems. There are many sophisticated methods available for a wide variety of problems. The general idea behind all those approaches is the same: Instead of explicitly stating the detailed way to solve a certain problem, we want the computer to *learn* it from given *data* (usually referred to as *training data*). One of the major problems in machine learning - the one we focus on in this paper - is the task of *classification*.

Definition 1.1 (Classification problem) *Given a set of labeled* training examples

$$X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^n \times \{1, ..., C\} \tag{1}$$

the classification problem *is the task of assigning a distinct label* $y \in \{1, ..., C\}$ *to the previously unseen data point* $\mathbf{x} \in \mathbb{R}^n$ *based on the* training set X.

Remark 1 (Notation) *We use the bold typeface to distinguish the i-th vector* $\mathbf{x}_i$ *from the i-th entry* x_i *of the vector* $\mathbf{x}$.

The study of the classification problem has a rich history in the field of machine learning. It dates back to the 1930s where Fisher developed a *linear discriminant function* in order to distinguish four kinds of flowers based on certain measurements [Fis36]. Many techniques for tackling the classification problem have been proposed since Fisher's paper. One of them were so called *binary decision trees* that were introduced by Hunt et al. in the 1960s [HMS66]. Binary decision trees are inductively trained classifiers that classify a data point $\mathbf{x}$ by subsequently thresholding a certain *feature* x_i[1]. Because of their simplicity and good performance, they were very popular for a long time. However, decision trees in their original form tend to *overfit*[2] the training data severely. The performance of trees could not match the performance of newly proposed classifiers like SVMs [Vap95, CV95]. Hence, for many classification tasks other methods were more favorable. Binary decision trees regained attention in 2001 when Breiman proposed the concept of so called *random forests* [Bre01]. Random forests are ensembles of randomly trained binary decision trees. Because of the randomness in training, the correlation between multiple trees in an ensemble decreases which in turn can increase the generalization performance [Bis06, Chapter 14.2]. Random forests are among the best performing classifiers today.
In this paper we discuss the concept of so called *decision jungles* which are ensembles of randomly trained *decision DAGs*. Decision jungles were introduced by Shotton et al. in 2013 [SSK+13a]. The concept is very similar to the one of random forests. However, the initial motivation was quite different: While Breiman introduced random forests in order to gain accuracy, Shotton et al. proposed decision jungles as a more memory efficient classifier that can be used for scenarios where little memory consumption is crucial (e.g. embedded systems). Their resulting concepts did not only reduce memory consumption tremendously, it also led to consistently better generalization performance. This makes decision jungles interesting not only for memory precious tasks, but for all classification tasks.

1.1 Outline and Contributions

In this paper we discuss decision jungles (ensembles of randomly trained decision DAGs) as proposed by Shotton at al. in [SSK+13a]. We start with a formal introduction where we compare the concept to that of binary decision trees. We then discuss training decision jungles where we focus on the LSEARCH optimization algorithm. In the last section we perform several experiments in order to (1) compare the performance of decision jungles; and random forests for the task of multi class classification (2) test the influence of different parameter settings on the classification accuracy as well as the training time.
Our contributions are: (1) We propose an efficient way of implementing the LSEARCH optimization algorithm; (2) we verify the claims by Shotton et al. that decision jungles yield better generalization performance than random forests while decreasing memory consumption; (3) we publish an open source implementation for training decision jungles.

[1] Decision trees can also be used for categorical data. See section 2.1 for details.

[2] A classifier that *overfits* a training set adapts strongly to every detail in the training set. Because data usually contains a certain amount of noise, those classifiers tend perform somewhat poorly on unseen data.

2 Formal Introduction

In this section we give a formal introduction to the topic. We start by giving formal definitions of decision DAGs and decision trees. We then proceed to compare the two ideas from a conceptual point of view.
Because decision DAGs can be seen as a generalization of binary decision trees, we first give the definition of decision DAGs and then specialize it to that of binary decision trees. By doing so, we can compare the two in one formal framework.

Definition 2.1 (Class histogram) *A* class histogram (short: histogram) *is a function* $h : \{1,...,C\} \mapsto \mathbb{R}$ *with* $\sum_{i=1}^{C} h(i) = 1$.

Definition 2.2 (decision DAG) *A* decision DAG *is a rooted DAG (directed acyclic graph)*[3] $G = (V,E)$ *for which all directed path from the root node r to a node v are of the same lengths and whose nodes* $v \in V$ *are augmented with the following attributes:*

- *If v is a leaf node, then v is associated with a class histogram* h_v
- *If v is not a leaf node (i.e. the root node or an internal node), then v is augmented with the tuple* $(d_v, \theta_v, l_v, r_v) \in \{1,...,n\} \times \mathbb{R} \times \{w \in V : (v,w) \in E\}^2$ *where*
  - d_v *is the* feature dimension
  - θ_v *is the* threshold
  - l_v *is the* left child node
  - r_v *is the* right child node

The additional requirement that all directed path from the root node r to a node v must have the same length makes sure that we have well defined *levels* that can be optimized during training.

Definition 2.3 (Binary decision tree) *A* binary decision tree *is a decision DAG* $G = (V,E)$ *whose nodes* $v \in V$ *have* $in(v) \leq 1$.

Remark 2 (Decision DAG training) Learning *or* training *a decision DAG refers to the task of finding* optimal *parameters* $(d_v, \theta_v, l_v, r_v)$ *for each node v. Learning a decision DAG is fundamentally more challenging than learning a binary decision tree because the parameters* l_v *and* r_v *(i.e. the graph structure) are fixed for trees. Therefore, only the threshold* θ_v *and the feature dimension* d_v *have to be optimized for trees. The* LSEARCH *optimization algorithm, which we discuss in section 3, optimizes the threshold and the graph structure simultaneously.*

In order to use decision DAGs (and decision trees) to solve the classification problem, we need to define a *classifier* $f_G : \mathbb{R}^n \mapsto \{1,...,C\}$ for a decision DAG G. Since we treat decision trees as a specialization of decision DAGs, we only need to define the classifier f_G for decision DAGs. Let $G = (V,E)$ be a rooted decision DAG. We inductively define a class histogram $\hat{h}_v : \mathbb{R}^n \mapsto (\{1,...,C\} \mapsto \mathbb{R})$ for every node v as follows:

- If v is a leaf node, then $\hat{h}_v(\mathbf{x}) = h_v$
- If v is not a leaf node, then

$$\hat{h}_v(\mathbf{x}) = \begin{cases} \hat{h}_{l_v} & \text{if} \quad x_{d_v} \leq \theta_v \\ \hat{h}_{r_v} & \text{if} \quad x_{d_v} > \theta_v \end{cases} \tag{2}$$

Hence, $\hat{h}_v(\mathbf{x})$ is the class histogram stored at the leaf node which is reached if you pass the data point $\mathbf{x}$ from v downwards according to the rules. The classifier f_G then is defined as

$$f_G : \mathbb{R}^n \mapsto \{1,...,C\}, \mathbf{x} \mapsto \underset{c=1,...,C}{\arg\max}(\hat{h}_r(\mathbf{x}))(c) \tag{3}$$

[3] See appendix A for a formal definition.

where $r \in V$ is the root node of G. Based on this formal framework, we can also define the *empirical decision confidence* p_G.

$$p_G : \mathbb{R}^n \mapsto \mathbb{R}, \mathbf{x} \mapsto \max_{c=1,\ldots,C} (\hat{h}_r(\mathbf{x}))(c) \tag{4}$$

Remark 3 *f_G is not well defined for all G because the* arg max *over the class histogram $\hat{h}$ might not be well defined. This is the case when there are multiple labels $c_1, \ldots, c_k$ with $\max_{c=1,\ldots,C}(\hat{h}_r(\mathbf{x}))(c) = (\hat{h}_r(\mathbf{x}))(c_1) = \ldots = (\hat{h}_r(\mathbf{x}))(c_k)$. In those cases, we choose the smallest class label c_i (i.e. $c_i < c_j, i \neq j$).*

The decision to pass a data point $\mathbf{x} \in \mathbb{R}^n$ at node $v \in V$ to the left child node instead of passing it to the right child node if $x_{d_v} = \theta_v$ is somewhat arbitrary.
Shotton et al. initially introduces the concept of decision DAGs as a memory efficient alternative to binary decision trees. Figure 1a and figure 1b show the same classifier. At the top the classifier is expressed in terms of a binary decision tree and at the bottom it is expressed as a decision DAG. This gives some intuition that decision DAGs indeed consume less memory than binary decision trees while maintaining roughly the same expressive power. In fact, the LSEARCH optimization algorithm they proposed trains a DAG *level-wise* and controls the *width* of each level by a *merging schedule*. Using an appropriate merging schedule (e.g. a bounded function), one can easily proof that decision DAGs learned using this algorithm consume less memory than ordinary binary decision trees.

Definition 2.4 (Level and width) *Let $G = (V, E)$ be a decision DAG. The l-th level of G is the set $\{v \in G : |path(r, v)| = l\}$. The* width *of the l-th level, is the number of nodes on the l-th level. The* width *of the decision DAG G is the maximum width of any level.*

Remark 4 *The definition of a level is well defined because we require that all directed paths from r to v have the same length.*

Let $s : \mathbb{N} \mapsto \mathbb{N}$ be a monotonic *merging schedule* that determines the width $s(l)$ of each level l. At each iteration of the algorithm, it generates $s(l+1)$ nodes and optimizes the parameters of all nodes on level l.

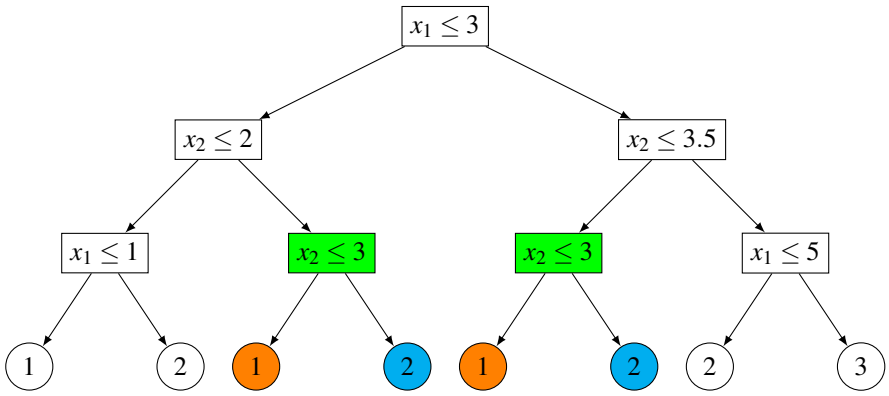
Lemma 1 *The number of nodes in a decision DAG G learned by* LSEARCH *optimization algorithm is bounded by $ds(d)$ where d is the depth of G.*

Proof: At each level l, $s(l+1)$ new nodes are generated. Since s is monotonic, a DAG of depth d has at most $ds(d)$ nodes, □
As training ordinary binary decision trees implies a merging schedule of the form $s(l) = 2^l$ in this framework, any function s that grows slower than 2^l leads to a DAG that consumes less memory. In fact, Shotton et al. use bounded merging schedules. Using a bounded merging schedule, the number of nodes grows linearly in the depth of the DAG.
So far, we only focused on single DAGs. We now discuss ensembles of *random DAGs*. *Random forests*, which are ensembles of random trees, were made popular by Breiman in 2001 who showed that combining random classifiers to an ensemble can boost performance significantly. Shotton et al. picked up his ideas and applied them to decision DAGs.

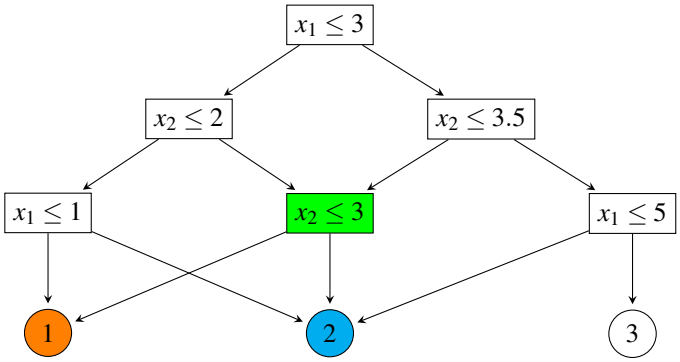
Definition 2.5 (Random decision DAG) *A* random decision DAG *is a decision DAG for which the attributes Θ_v of each internal node $v \in V$ are independently and identically distributed random variables.*

By letting the attributes of a random decision DAG G be random variables, the resulting classifier f_G is also a random variable. If we sample multiple random decision DAG (i.e. *learn* multiple decision DAGs with randomness), we get an *ensemble of random classifiers*. Shotton et al. named those ensembles *decision jungles*.

Definition 2.6 (Decision jungles) *A* decision jungle *$J = (G_1, \ldots, G_m)$ is an ensemble of m random decision DAGs $G_1, \ldots, G_m$.*



(a) A binary decision tree



(b) A decision DAG

Figure 1: Figures 1a and 1b show the same classifier for a two-dimensional three class classification problem. The major difference between the two models is that in the DAG nodes may have more than one parent node. The leaf nodes (marked by circles) show the class label that is assigned to a data point that is passed to the node (i.e. the $\arg\max \hat{h}_v$).

There are many ways to define the classifier that is induced by an ensemble of multiple base classifiers. The most popular way, which is also the method Breiman used in random forests, is a voting scheme. The data point that is supposed to be classified is passed to each DAG individually. Each DAG then votes for the class label it predicts for the data point. Finally, the class which received the highest number of votes is chosen as the prediction result for the entire ensemble. Formally, we can define a classifier f_J for an ensemble $J = (G_1, ..., G_m)$ as follows.

$$f_J : \mathbb{R}^n \mapsto \{1, ..., C\}, \mathbf{x} \mapsto \underset{c=1,...,C}{\arg\max} \sum_{i=1}^{m} \mathbb{1}(c, f_{G_i}(\mathbf{x})) \tag{5}$$

where $\mathbb{1}$ is the indicator function:

$$\mathbb{1}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Let $r_1, ..., r_m$ be the root nodes of $G_1, ..., G_m$. Analog to the case of a single DAG, we can define the empirical decision confidence for ensembles.

$$p_J : \mathbb{R}^n \mapsto \mathbb{R}, \mathbf{x} \mapsto \max_{c=1,...,C} \frac{1}{m} \sum_{i=1}^{m} (\hat{h}_{r_i}(\mathbf{x}))(c) \tag{7}$$

Again, we achieve well-definedness of f_J by always selecting the smallest c that gives rise to the maximum. One might wonder why we chose to consider ensembles of random decision DAGs rather than deterministically trained *optimal* (in terms of some error function) decision DAGs. There essentially are two reasons.

Firstly, if we considered deterministically trained DAGs on the same training set, we would end up with m copies of the same classifier in our ensemble. Secondly - and more importantly - Breiman showed that adding more random classifiers to an ensemble is unlikely to cause overfitting [Bre01, Theorem 1.2] and a better *generalization performance* can be achieved if the individual classifiers are only weakly correlated [Bre01, Theorem 2.3].

2.1 Categorical Data

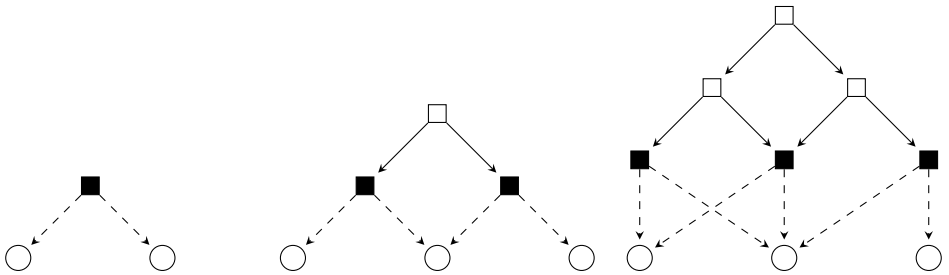
So far, we assumed our data points $\mathbf{x}$ to be real valued vectors. However, in practice one often faces *categorical data*. The feature domain of a categorical data point $\mathbf{x}$ is a set of discrete labels that do not exhibit any algebraic structure. Fortunately, by using a simple *feature map* we can also handle categorical data within our framework.
Let $\times_{i=1}^{n} \Omega_i$ be the feature domain of a categorical classification problem. The feature map ϕ is defined as

$$\phi : \bigtimes_{i=1}^{n} \Omega_i \mapsto \bigtimes_{i=1}^{n} \mathbb{R}^{|\Omega_i|}, (x_1, ..., x_n) \mapsto (\mathbb{1}(x_1, \omega_{1,1}), ..., \mathbb{1}(x_1, \omega_{1,|\Omega_1|}), ..., \mathbb{1}(x_n, \omega_{n,1}), ..., \mathbb{1}(x_n, \omega_{n,|\Omega_n|})) \quad (8)$$

where $\Omega_i = \{\omega_{i,1}, ..., \omega_{i,k}\}$. We use this feature map for our experiments on the CONNECT 4 data set in section 5.4.

Example 1 *Let $\Omega_1 = \{hot, cold\}$ and $\Omega_2 = \{winter, spring, summer, fall\}$ be the domain of a categorical classification problem. The data point $\mathbf{x} = (hot, spring) \in \Omega_1 \times \Omega_2$ has the following image under the feature map*

$$\phi(hot, spring) = (\underbrace{1,0}_{\Omega_1}, \underbrace{0,1,0,0}_{\Omega_2}) \quad (9)$$



(a) First iteration (b) Second iteration (c) Third iteration

Figure 2: At each iteration of the algorithm, a new level of nodes is added to the graph and the attributes of the former leaf nodes have be determined. The figures above show the first three iterations. Rectangular nodes represent internal nodes and circular ones represent leaf nodes. The filled nodes are the ones whose parameters have to be determined.

3 Training

In this section we present the training algorithm proposed by Shotton et al. for learning random decision DAGs. However, in contrast to their paper, we formulate the algorithm in a computationally efficient way. We first discuss training a single random decision DAG and then proceed to learning random ensembles.

Traditionally, decision trees are learned by subsequently *splitting* nodes by *thresholding* a single *feature*[4].

To put learning random decision DAG into a more general framework, Shotton et al. formulated the problem in terms of minimizing an *objective function*[5]. The objective function gives a score for all parameter settings. Hence, minimizing an objective function means, that the parameters have to be determined that give rise to the smallest score. If the objective function is not convex[6], finding these optimal parameters can be difficult. In those cases, many training algorithms only attempt to find local minima.

Like ordinary tree training algorithms, Shotton's method for learning a random decision DAG also works by subsequently splitting nodes. However, more parameters have to be learned for decision DAGs than for decision trees. More specifically, the left and right child nodes for each node have to be learned for DAGs. Those parameters are fixed for trees due to the graph structure.

The algorithm learns a decision DAG *level-by-level*. At each iteration, it adds another level of nodes to the decision DAG and determines the optimal attributes of the former leaf nodes, which now are internal nodes. Figure 2 illustrates the concept.

In order to discuss the algorithm formally, we first need to introduce some notation. Let $G = (V,E)$ be a decision DAG and let $v \in V$ be a node in the DAG. We denote the training examples at v (i.e. the training examples from the original training set that have been passed to v from its parents) by S_v. At each iteration, we call the former level of leaf nodes *parent level* and the newly added level of nodes *child level*. Analog, we call the individual nodes *parent nodes* and *child nodes* respectively. The objective function is defined in terms of the *class entropy* at the child level.

Definition 3.1 (Entropy) *Let $h : \{1,...,C\} \mapsto \mathbb{R}$ be a histogram. The entropy $H(h)$ is defined as*

$$H : (\{1,...,C\} \mapsto \mathbb{R}) \mapsto \mathbb{R}, h \mapsto -\sum_{i=1}^{C} h(i) \log_2(h(i)) \tag{10}$$

We can easily extend the histogram-based definition of the entropy to training sets. Let $X \subset \mathbb{R}^n \times \{1,...,C\}$ be a training set. The entropy of X is defined as

$$H(X) \equiv H(h_X) \tag{11}$$

[4] In this context, the term *feature* refers to a certain dimension of the feature vector.

[5] Objective functions are often called *error functions* or *energy functions.*

[6] A convex function has a unique global minimum.

where h_X is defined by counting class frequencies

$$h_X : \{1,...,C\} \mapsto \mathbb{R}, c \mapsto \frac{|\{(\mathbf{x},y) \in X : y = c\}|}{|X|} \tag{12}$$

Let $p_1,...,p_k \in V$ be the parent nodes and let $c_1,...,c_l \in V$ be the child nodes. The objective function E, which we seek to minimize, is defined as

$$E(\Theta_{p_1},...,\Theta_{p_k}) = \sum_{i=1}^{l} |S_{c_i}| H(S_{c_i}) \tag{13}$$

where $\Theta_{p_i} = (d_{p_i}, \theta_{p_i}, l_{p_i}, r_{p_i})$ are the attributes of the parent nodes as defined in definition 2.2. Hence, the objective function is just a weighted entropy. The information gain impurity measure for ordinary tree training is a special case of this function. In this sense it can be understood as a natural generalization of the learning problem for trees.
It might not be obvious how the energy function depends on the attributes of the parent nodes, which ought to be optimized. However, this becomes apparent as soon as we look at the definition of S_{c_i} for a child node c_i. S_{c_i} is the set of all training examples that are passed from its parents to c_i; that is

$$S_{c_i} = \bigcup_{j=1,..,k \,:\, l_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : x_{d_{p_j}} \leq \theta_{p_j}\} \cup \bigcup_{j=1,..,k \,:\, r_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : x_{d_{p_j}} > \theta_{p_j}\} \tag{14}$$

From this formulation we see that all S_{c_i} - and therefore E - directly depend on the parameter settings of the parent nodes.
We now present the LSEARCH optimization algorithm as it was proposed by Shotton at al. The algorithm is designed to find a local minimum of the objective function. It optimizes each level of the DAG individually. Besides the LSEARCH algorithm, they also proposed an alternative algorithm called CLUSTERSEARCH. However, they say that it performs consistently worse than the LSEARCH algorithm. For this reason, we only focus on the LSEARCH algorithm in this paper.
The algorithm works as follows: Given an initialization of all parameters $\Theta_{p_1},...,\Theta_{p_k}$, the algorithm greedily optimizes a single parameter while leaving all the other ones fixed. Algorithm 1 depicts the optimization procedure.

Algorithm 1 LSEARCH optimization algorithm [SSK+13a]. The notation is slightly adjusted to simplify further derivations.

function LSEARCH($\Theta_{p_1},...,\Theta_{p_k}$)
  while something changes **do**
    for $i = 1,...,k$ **do**
      $\mathcal{F} \leftarrow$ random feature selection
      $(d_{p_i}, \theta_{p_i}) \leftarrow \arg\min_{d \in \mathcal{F}, \theta \in R} E(\Theta_{p_1},...,\Theta_{p_{i-1}},(d,\theta,l_{p_i},r_{p_i}),\Theta_{p_{i+1}},...,\Theta_{p_k})$
    end for
    for $i = 1,...,k$ **do**
      $l_{p_i} \leftarrow \arg\min_{l=c_1,...,c_l} E(\Theta_{p_1},...,\Theta_{p_{i-1}},(d_{p_i},\theta_{p_i},l,r_{p_i}),\Theta_{p_{i+1}},...,\Theta_{p_k})$
      $r_{p_i} \leftarrow \arg\min_{r=c_1,...,c_l} E(\Theta_{p_1},...,\Theta_{p_{i-1}},(d_{p_i},\theta_{p_i},l_{p_i},r),\Theta_{p_{i+1}},...,\Theta_{p_k})$
    end for
  end while
  return $\Theta_{p_1},...,\Theta_{p_k}$
end function

Remark 5 *From the original paper it is not entirely clear how they optimize the threshold. They state it "is done by random sampling in a manner similar to decision forest training" [SSK+13a]. There are two popular approaches for training random forests: (1) Sample a random feature and find the optimal threshold (2) Sample a random feature and a random threshold . From the supplementary material it seems like they went for the second option for their experiments [SSK+13b]. However, our implementation follows the first approach that is also used by Piotr Dollár [Dol].*

Algorithm 2 Algorithm for learning a decision DAG using the LSEARCH optimization algorithm. The DAG is learned level-by-level. After each iteration a new level of $s(l)$ nodes is added and the parameters of the former leaf nodes are optimized by the LSEARCH algorithm. s a merging schedule as introduced in section 2.

1: $k \leftarrow 1$
2: **for** l = 1,2,.. **do**
3: Add $s(l)$ new leaf nodes $c_1, ..., c_{s(l)}$
4: initialize $\Theta_{p_1}, ..., \Theta_{p_k}$
5: $\Theta_{p_1}, ..., \Theta_{p_k} \leftarrow \text{LSEARCH}(\Theta_{p_1}, ..., \Theta_{p_k})$
6: $k \leftarrow s(l)$
7: **end for**

Algorithm 2 shows how the LSEARCH optimization algorithm can be used to learn an entire DAG. Before we discuss implementation details, we shortly analyze the algorithm using the following two propositions.

Theorem 3.2 *The* LSEARCH *algorithm terminates.*

Proof: For each parent node and each feature, there is only a finite number of intervals of possible thresholds for which the objective function takes on different values while leaving all other parameters fixed. Therefore, there is only a finite number of configurations for $\Theta_{p_1}, ..., \Theta_{p_k}$. Hence, if the algorithm did not terminate, it would cycle through a sequence of configurations. Let $\gamma_1, ..., \gamma_r$ be such a sequence of configurations. (i.e. $\text{LSEARCH}(\gamma_i) = \gamma_{i+1}$, for $i = 1, ..., r-1$ and $\text{LSEARCH}(\gamma_r) = \gamma_1$). Because a parameter changes if and only if the objective function decreases, the following must hold

$$E(\gamma_1) > E(\gamma_2) > \ldots > E(\gamma_{r-1}) > E(\gamma_r) \tag{15}$$

Since the algorithm cycles through these configurations, it must also hold that

$$E(\gamma_r) > E(\gamma_1) \tag{16}$$

Otherwise, it would not perform the cycle. But this implies

$$E(\gamma_1) > E(\gamma_1) \tag{17}$$

This is a contradiction. Hence, the algorithm terminates. □
Theorem 3.2 directly implies the following theorem.

Theorem 3.3 *The* LSEARCH *algorithm converges to a local minimum of the objective function in a finite number of steps.*

3.1 Implementation Details

In this section we discuss implementation details of the LSEARCH algorithm. Those details have been omitted from the original paper and therefore form the main contribution of this paper. Efficiently implementing the algorithm is not trivial. This is mostly due to the fact that evaluating the objective function is computationally expensive: Given only the parameter settings of the parent node, the training examples first have to be assigned to the child nodes and then the entropies at the child nodes have to be calculated. We now derive efficient methods for evaluating the objective function for the different optimization steps. Our key idea is to maintain two histograms $h^l_{p_i}$ and $h^r_{p_i}$ for each parent node p_i. Those histograms capture the class distribution at the left and right child node if p_i was the only parent node of the latter. We call them left and right *virtual histogram* of p_i. Formally, let p_i be a parent node we define $h^l_{p_i}$ and $h^r_{p_i}$ as follows

$$h^l_{p_i} : \{1, ..., C\} \mapsto \mathbb{R},\ h^l_{p_i}(i) = \frac{|\{(\mathbf{x}, y) \in S_{p_i} : y = i \wedge x_{d_{p_i}} \leq \theta_{p_i}\}|}{|S_{p_i}|} =: \frac{\overline{h}^l_{p_i}(i)}{|\overline{h}^l_{p_i}|} \tag{18}$$

$$h^r_{p_i} : \{1, ..., C\} \mapsto \mathbb{R},\ h^r_{p_i}(i) = \frac{|\{(\mathbf{x}, y) \in S_{p_i} : y = i \wedge x_{d_{p_i}} > \theta_{p_i}\}|}{|S_{p_i}|} =: \frac{\overline{h}^r_{p_i}(i)}{|\overline{h}^r_{p_i}|} \tag{19}$$

We store those virtual histograms for each parent node p_i using $C+1$ values per histogram (C numerators $\overline{h}(i)$ and 1 denominator $|\overline{h}|$). This allows us to express the entropy at the child nodes using the virtual histograms.

Corollary 1 *Let c_i be a child node. It holds*

$$H(S_{c_i}) = H\left(\sum_{j=1,\ldots,k \,:\, l_{p_j}=c_i} h^l_{p_j} + \sum_{j=1,\ldots,k \,:\, r_{p_j}=c_i} h^r_{p_j}\right) \tag{20}$$

where for two virtual histograms h_1 and h_2 we define

$$(h_1+h_2)(i) := \frac{\overline{h}_1(i)+\overline{h}_2(i)}{|\overline{h}_1|+|\overline{h}_2|} \tag{21}$$

Proof: The entropy is defined as

$$H(S_{c_i}) = H(h_{S_{c_i}}) \tag{22}$$

From equation 14 we have

$$S_{c_i} = \bigcup_{j=1,..,k \,:\, l_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : x_{d_{p_j}} \leq \theta_{p_j}\} \cup \bigcup_{j=1,..,k \,:\, r_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : x_{d_{p_j}} > \theta_{p_j}\} \tag{23}$$

and therefore we get for $r \in \{1,\ldots,C\}$

$$\begin{aligned}
h_{S_{c_i}}(r) &= \frac{|\{(\mathbf{x},y) \in S_{c_i} : y = r\}|}{|S_{c_i}|} \\
&= \frac{|\bigcup_{j=1,..,k \,:\, l_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} \leq \theta_{p_j}\} \cup \bigcup_{j=1,..,k \,:\, r_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} > \theta_{p_j}\}|}{|S_{c_i}|} \\
&= \frac{|\bigcup_{j=1,..,k \,:\, l_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} \leq \theta_{p_j}\} \cup \bigcup_{j=1,..,k \,:\, r_{p_j}=c_i} \{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} > \theta_{p_j}\}|}{\sum_{j=1,..,k \,:\, l_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : \wedge x_{d_{p_j}} \leq \theta_{p_j}\}| + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} > \theta_{p_j}\}|} \\
&= \frac{\sum_{j=1,..,k \,:\, l_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} \leq \theta_{p_j}\}| + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} > \theta_{p_j}\}|}{\sum_{j=1,..,k \,:\, l_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : \wedge x_{d_{p_j}} \leq \theta_{p_j}\}| + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} |\{(\mathbf{x},y) \in S_{p_j} : y = r \wedge x_{d_{p_j}} > \theta_{p_j}\}|} \\
&= \frac{\sum_{j=1,..,k \,:\, l_{p_j}=c_i} \overline{h}^l_{p_j}(r) + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} \overline{h}^r_{p_j}(r)}{\sum_{j=1,..,k \,:\, l_{p_j}=c_i} |\overline{h}^l_{p_j}| + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} |\overline{h}^r_{p_j}|} = \left(\sum_{j=1,..,k \,:\, l_{p_j}=c_i} h^l_{p_j} + \sum_{j=1,..,k \,:\, r_{p_j}=c_i} h^r_{p_j}\right)(r)
\end{aligned}$$

□

We now present efficient methods for finding the optimal threshold and feature as well as the optimal child nodes.

3.1.1 Optimizing the Feature and Threshold

As stated in remark 5, we only inject randomness by selecting random features; we do not select a random threshold. Our goal is to execute line 5 of algorithm 1 efficiently. The main challenge in this task is the optimization of the threshold given a fixed feature. For this purpose, we adapted the idea Piotr Dollár used for his implementation of random forests [Dol] to our case. We explain the concept in full detail. In the following we assume a fixed parent node p_i with fixed child nodes l_{p_i} and r_{p_i}.

Let d be a fixed feature dimension. If we sort S_{p_i} according to the d-th entry of all vectors, we only have to evaluate the objective function using a single threshold for every two subsequent points. Figure 3 illustrates the idea. For every interval, we select the middle value between the two data points. Algorithm 3 depicts the preliminary form of the optimization algorithm.

The following derivations show how we can evaluate the objective function at each iteration in constant time using the virtual histograms. First, we can neglect the error contribution of all child nodes other than l_{p_i} and r_{p_i} because it is the same for all iterations.

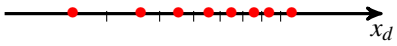


Figure 3: The red points represent data points sorted according to one feature dimension. On every interval defined by two subsequent data points, the objective function is constant. Hence, we only have to evaluate the objective function for one threshold per interval.

Algorithm 3 Preliminary form of the threshold and feature optimization algorithm. $\mathcal{F}$ is a random feature selection.

for $d \in \mathcal{F}$ **do**
  Sort S_{p_i} according to d-the feature dimension
  for $j = 1, ..., |S_{p_i}| - 1$ **do**
    $\theta \leftarrow \frac{(\mathbf{x}_j)_d + (\mathbf{x}_{j+1})_d}{2}$
    if $E(\Theta_{p_1}, ..., \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, ..., \Theta_{p_k}) < E(\Theta_{p_1}, ..., \Theta_{p_k})$ **then**
      $d_{p_i} \leftarrow d$
      $\theta_{p_i} \leftarrow \theta$
    end if
  end for
end for

Corollary 2 *Let p_i be a parent node and let $\mathcal{F}$ be a random feature selection. It holds*

$$\underset{d \in \mathcal{F}, \theta \in R}{\arg\min}\, E(\Theta_{p_1}, ..., \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, ..., \Theta_{p_k}) = \underset{d \in \mathcal{F}, \theta \in R}{\arg\min} \underbrace{|S_{l_{p_i}}| H(S_{l_{p_i}}) + |S_{r_{p_i}}| H(S_{r_{p_i}})}_{=:\tilde{E}} \tag{24}$$

Furthermore, $|S_{l_{p_i}}| H(S_{l_{p_i}})$ and $|S_{r_{p_i}}| H(S_{r_{p_i}})$ can efficiently be computed using virtual histograms as stated in corollary 1. We also note that only $h^l_{p_i}$ and $h^r_{p_i}$ change during the algorithm. Hence, for each of the two child nodes we can precompute a virtual histogram h^l and h^r that represent the contributions from all other parent nodes of the respective child nodes. That is

$$h^l = \sum_{j=1,...,k \,:\, l_{p_j} = l_{p_i}, j \neq i} h^l_{p_j} + \sum_{j=1,...,k \,:\, r_{p_j} = l_{p_i}, j \neq i} h^r_{p_j} \tag{25}$$

$$h^r = \sum_{j=1,...,k \,:\, l_{p_j} = r_{p_i}, j \neq i} h^l_{p_j} + \sum_{j=1,...,k \,:\, r_{p_j} = r_{p_i}, j \neq i} h^r_{p_j} \tag{26}$$

Using those precomputed virtual histograms, we can evaluate the objective function as follows

$$\tilde{E}(h^l + h^l_{p_i}, h^r + h^r_{p_i}) = |S_{l_{p_i}}| H(S_{l_{p_i}}) + |S_{r_{p_i}}| H(S_{r_{p_i}}) \tag{27}$$

$$= (|\overline{h}^l| + |\overline{h}^l_{p_i}|) H(h^l + h^l_{p_i}) + (|\overline{h}^r| + |\overline{h}^r_{p_i}|) H(h^r + h^r_{p_i}) \tag{28}$$

Now, instead of computing the virtual histograms at each iteration by assigning each training example in S_{p_i} to either $S_{r_{p_i}}$ or $S_{l_{p_i}}$ individually, we just update the histograms from the previous iteration. This is possible because at each iteration only one training example moves from the $S_{r_{p_i}}$ to $S_{l_{p_i}}$ [7].

We can further reduce the runtime by subsequently updating $(|\overline{h}^l| + |\overline{h}^l_{p_i}|) H(h^l + h^l_{p_i})$ and $(|\overline{h}^r| + |\overline{h}^r_{p_i}|) H(h^r + h^r_{p_i})$. The following corollary expresses the idea how to update the entropy in constant time if only one bin of the histogram changes.

Corollary 3 *Let h be a virtual histogram. It holds*

$$|\overline{h}| H(h) = \sum_{i=1}^{C} -\overline{h}(i) \log_2(\overline{h}(i)) + |\overline{h}| \log_2(|\overline{h}|) \tag{29}$$

[7] There is one exception to this rule: If several examples have the same feature values, then all those examples would move. We can easily avoid issues caused by this exception by only accepting thresholds for which $(\mathbf{x}_i)_d \neq (\mathbf{x}_{i+1})_d$.

Proof:

$$|\overline{h}|H(h) = -|\overline{h}|\sum_{i=1}^{C} h(i)\log_2(h(i)) \tag{30}$$

$$= |\overline{h}|\left(-\sum_{i=1}^{C}\frac{\overline{h}(i)}{|\overline{h}|}\log_2\left(\frac{\overline{h}(i)}{|\overline{h}|}\right)\right) \tag{31}$$

$$= -\sum_{i=1}^{C}\overline{h}(i)\log_2\left(\frac{\overline{h}(i)}{|\overline{h}|}\right) \tag{32}$$

$$= -\sum_{i=1}^{C}\overline{h}(i)(\log_2(\overline{h}(i)) - \log_2(|\overline{h}|)) \tag{33}$$

$$= \sum_{i=1}^{C} -\overline{h}(i)\log_2(\overline{h}(i)) + \log_2(|\overline{h}|)\underbrace{\sum_{i=1}^{C}\overline{h}(i)}_{|\overline{h}|} \tag{34}$$

□

Algorithm 4 shows the final optimization algorithm.

Algorithm 4 Final form of the threshold and feature optimization algorithm. $\mathcal{F}$ is a random feature selection.

```
1:  Precompute h^l and h^r
2:  b ← Ẽ(h^l_{p_i}, h^r_{p_i})
3:  for d ∈ F do
4:      Sort S_{p_i} according to d
5:      h^l_{p_i} ← h^l
6:      h^r_{p_i} ← h_{S_{p_i}} + h^r
7:      e_l ← 0
8:      e_r ← |h̄^r_{p_i}| H(h^r_{p_i})
9:      for j = 1, ..., |S_{p_i}| − 1 do
10:         θ ← ((x_j)_d + (x_{j+1})_d) / 2
11:         e_l ← e_l + |h̄^l_{p_i}| log_2(|h̄^l_{p_i}|) − h̄^l_{p_i}(y_j) log_2(h̄^l_{p_i}(y_j))
12:         e_r ← e_r + |h̄^r_{p_i}| log_2(|h̄^r_{p_i}|) − h̄^r_{p_i}(y_j) log_2(h̄^r_{p_i}(y_j))
13:         h̄^l_{p_i}(y_j) ← h̄^l_{p_i}(y_j) + 1
14:         h̄^r_{p_i}(y_j) ← h̄^r_{p_i}(y_j) − 1
15:         e_l ← e_l − |h̄^l_{p_i}| log_2(|h̄^l_{p_i}|) + h̄^l_{p_i}(y_j) log_2(h̄^l_{p_i}(y_j))
16:         e_r ← e_r − |h̄^r_{p_i}| log_2(|h̄^r_{p_i}|) + h̄^r_{p_i}(y_j) log_2(h̄^r_{p_i}(y_j))
17:         if e_l + e_r < b AND (x_j)_d ≠ (x_{j+1})_d then
18:             d_{p_i} ← d
19:             θ_{p_i} ← θ
20:             b ← e_l + e_r
21:         end if
22:     end for
23: end for
```

3.1.2 Optimizing the Child Nodes

In lines 8 and 9 of algorithm 1 the left and right child nodes of a parent node p_i are optimized. As before, naively evaluating the objective function is too expensive. In order to implement the optimization efficiently, we follow the same idea as before and use virtual histograms.

Let p_i be a fixed parent node and let $c_1, ..., c_l$ be the child nodes. For each child node c_i, we precompute a histogram that is the sum of c_j's parents' virtual histograms where we leave out p_i. That is

$$h^{c_i} = \sum_{j=1,...,k \,:\, l_{p_j}=l_{p_i}, j\neq i} h^l_{p_j} + \sum_{j=1,...,k \,:\, r_{p_j}=l_{p_i}, j\neq i} h^r_{p_j} \tag{35}$$

Using these histograms, we can efficiently evaluate the objective function for the case when only the child nodes l_{p_i} and r_{p_i} change.

$$E(l_{p_i}, r_{p_i}) = (|\overline{h}^{l_{p_i}}| + |\overline{h}^l_{p_i}|)H(h^{l_{p_i}} + h^l_{p_i}) + (|\overline{h}^r| + |\overline{h}^{r_{p_i}}_{p_i}|)H(h^{r_{p_i}} + h^r_{p_i}) + \sum_{j=1,...,l : c_j \notin \{l_{p_i}, r_{p_i}\}} |\overline{h}^{c_j}|H(h^{c_j}) \tag{36}$$

We can further reduce runtime by precomputing $|\overline{h}^{c_j}|H(h^{c_j})$ for all child nodes.

Remark 6 (Pure nodes) *We say a parent node p_i is* pure[8] *iff $H(S_{p_i}) = 0$. In ordinary tree training, pure nodes are not split any further. However, from the supplementary material [SSK+13b] one can tell, that all paths from the root node to all leaf nodes have the same length. In order to achieve this, pure nodes must be split further. Since there is no way to define a reasonable threshold if all training examples are of the same class, we found that choosing an arbitrary threshold and optimizing the child nodes of p_i such that $l_{p_i} = r_{p_i}$ results in a good performance.*

3.1.3 Initialization

In line 4 of algorithm 2 the parameters of the parent nodes $\Theta_{p_1}, ..., \Theta_{p_k}$ ought to be initialized. While this can be done randomly, Shotton et al. proposed a way to initialize the child nodes deterministically. This initialization yields a faster convergence speed as we verified in our experiments. The initialization works as follows: Sort the parent nodes according to the entropy of their training sets (i.e. sort according to $H(S_{p_i})$) and then assign the the child nodes such that parent nodes with high entropy do not have common child nodes.

3.1.4 Training Decision Jungles

Until now, we focused on training a single random decision DAG. However, decision jungles are ensembles of multiple random DAGs. There are two popular ways for training random ensembles of classifiers: (1) Train multiple random classifiers on the entire training set (2) Train each random classifier on a random selection of training examples . The latter approach is known as *bagging* and has been used by Breiman for his random forests [Bre01]. Unfortunately, Shotton et al. do not make clear whether or not they use bagging. For this reason, we implemented both methods and leave the choice to user.

3.1.5 Parameters

We now have presented all implementation details. In order to implement the entire algorithm, one needs to introduce some parameters. All of the parameters below are also adjustable in our implementation [Poh].

- $L \in \mathbb{N}$: Maximum number of levels to train
- $I \in \mathbb{N}$: Although the LSEARCH optimization algorithm eventually terminates, it may take many iterations of the outer `while`-loop to do so. In practice, however, it yields reasonably good parameter values after a small number of iterations. For this reason, we bound the number of iterations by I. This has also been done by Shotton et al. as they state in the supplementary material [SSK+13b].
- $F \in \mathbb{N}$: The number of features that are randomly sampled for optimizing the threshold
- $B \in \{\texttt{true}, \texttt{false}\}$: Whether or not bagging shall be used for training the ensemble
- $N_1 \in \mathbb{N}$: The number of training examples that are sampled per DAG with replacement when bagging is used.
- $M \in \mathbb{N}$: The total number of DAGs to train.

[8] In other words: A node p_i is pure if all its training examples at p_i have the same class label.

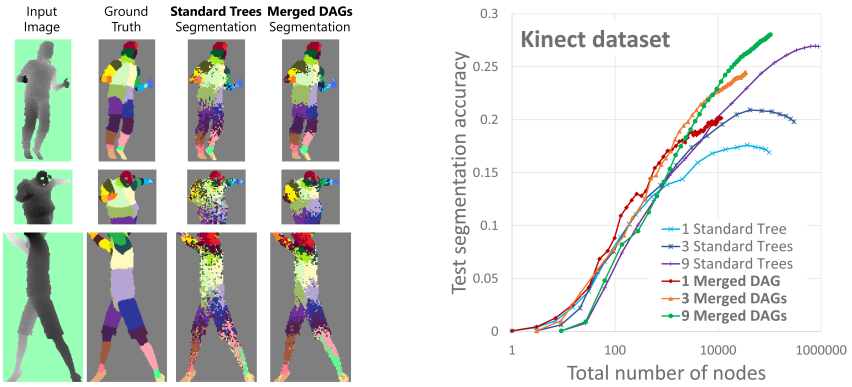


Figure 4: The results obtained for the task of semantic segmentation on the KINECT data set [SGF+12]. The left images shows the qualitative results whereas the right chart shows the quantitative results. Image source: Shotton et al. [SSK+13a].

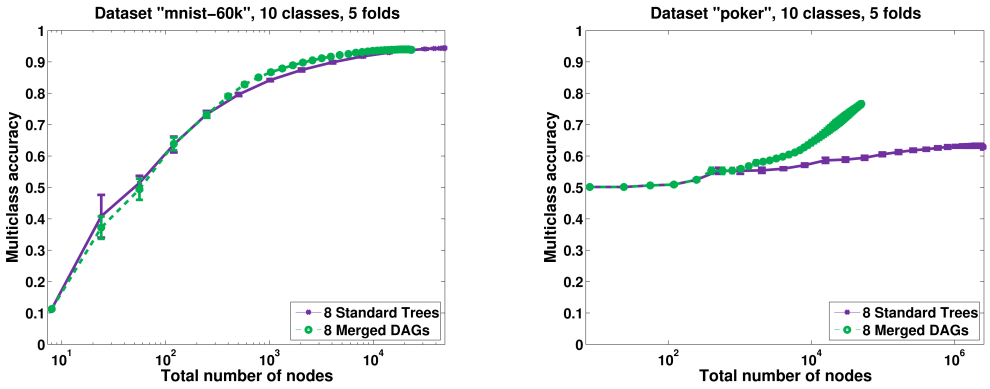


Figure 5: The results obtained for the task of multiclass classification on the MINST and POKER data sets. Image source: Shotton et al. [SSK+13a].

4 Results

Shotton et al. evaluated the performance of decision jungles for two tasks. The first tasks was multi class classification as it is defined in definition 1.1. The second one was the task of *semantic segmentation*.

Definition 4.1 (Discrete image) *A function* $I : \Lambda \mapsto V$ *is called a* discrete image *on the* grid $\Lambda = \{1,...,w\} \times \{1,...,h\}$.

Definition 4.2 (Semantic segmentation) *Let*

$$X = \{(I_i, Y_i) : i = 1,...,N, I_i : \Lambda \mapsto \mathbb{R}^n, Y_i : \Lambda \mapsto \{1,...,C\}\} \tag{37}$$

be a training set comprised of training images I_i *and* pixel maps Y_i. Semantic segmentation *is the task of finding a pixel map* $Y : \Lambda \mapsto \{1,...,C\}$ *for a previously unseen image* $I : \Lambda \mapsto \mathbb{R}^n$.

We can easily compute a semantic segmentation of an image by applying a classifier to each pixel individually. However, by doing so, we do not take into account the spatial layout of the segments in the image. For this reason, many state-of-the-art approaches use such a pixel-wise classification scheme as preprocessing step for more sophisticated methods (e.g. [HFL14]).
Figure 4 reports qualitative and quantitative results for the task of semantic segmentation on the KINECT data sets. There are two notable observations: First, the segmentation accuracy of decision jungles as measured by per-pixel error is significantly higher than the segmentation accuracy achieved with random forests. Second, the qualitative segmentation shows that the individual segments are spatially more coherent

Figure 6: A selection of figures from the MNIST data set.

when compared to the other approach. This experiment suggests that decision jungles could be favorable over random forests for semantic segmentation.
Two exemplary results for the task of multiclass classification are reported in figure 5. Again, decision jungles outperform random forests in terms of classification accuracy as well as model size. Especially on the Poker data sets decision jungles perform significantly better. More results on other data sets can be found in the supplemental material [SSK+13b].

5 Experiments

In this section we perform various experiments on the MNIST data set[9]. The data set is composed of 70,000 gray-scale images of handwritten digits. Each image has 28×28 pixels, which results in a total 784 feature dimensions. The set is split into a training set comprising 60,000 images and a test set comprising 10,000 images. Figure 6 shows a selection of examples. All experiments have been conducted with the following parameter settings:

- merging schedule $s(l) = \min(2^l, 128)$
- 28 feature samples per threshold optimization

All of the presented results are obtained using our LibJungle C++ library [Poh]. We used Piotr Dollár's MATLAB tool box [Dol] for comparing decision jungles and random forests.

5.1 Experiment 1: Maximum Number of Iterations

In a real world implementation, the number of iterations of the outer `while`-loop in the LSEARCH optimization algorithm[10] is bounded by a constant I. This experiments aims to investigate the influence of the iteration bound on the the following measurements:

- Test accuracy
- Depth of the resulting DAG (i.e. model size and evaluation time)
- Training time
- Convergence speed (Measured by the training error)

For each setting of I, we performed the experiment five times. Figure 7 reports the average measurements as well as the standard deviations.
The depth of the resulting DAGs is not only interesting because it indicates the model size, but also because it directly correlates with the evaluation time. In many applications, binary decision trees are chosen because their evaluation time is sub-linear in the number of feature dimensions.
The experiment shows that there is no significant gain in accuracy when the number of iterations is increased. The resulting DAGs perform equally well on the test set. However, increasing the iteration limit reduces the number of levels that have to be trained. Therefore, a high iterations limit results in models that are smaller and evaluate faster.
This experiment also verifies the claim by Shotton et al. that decision DAGs generalize better than decision trees. The gain in test accuracy is approximately 3.89%. However, as noted by them, this gain comes at the cost of twice the number of feature evaluations.

[9] Available under `http://yann.lecun.com/exdb/mnist/`.
[10] Algorithm 1

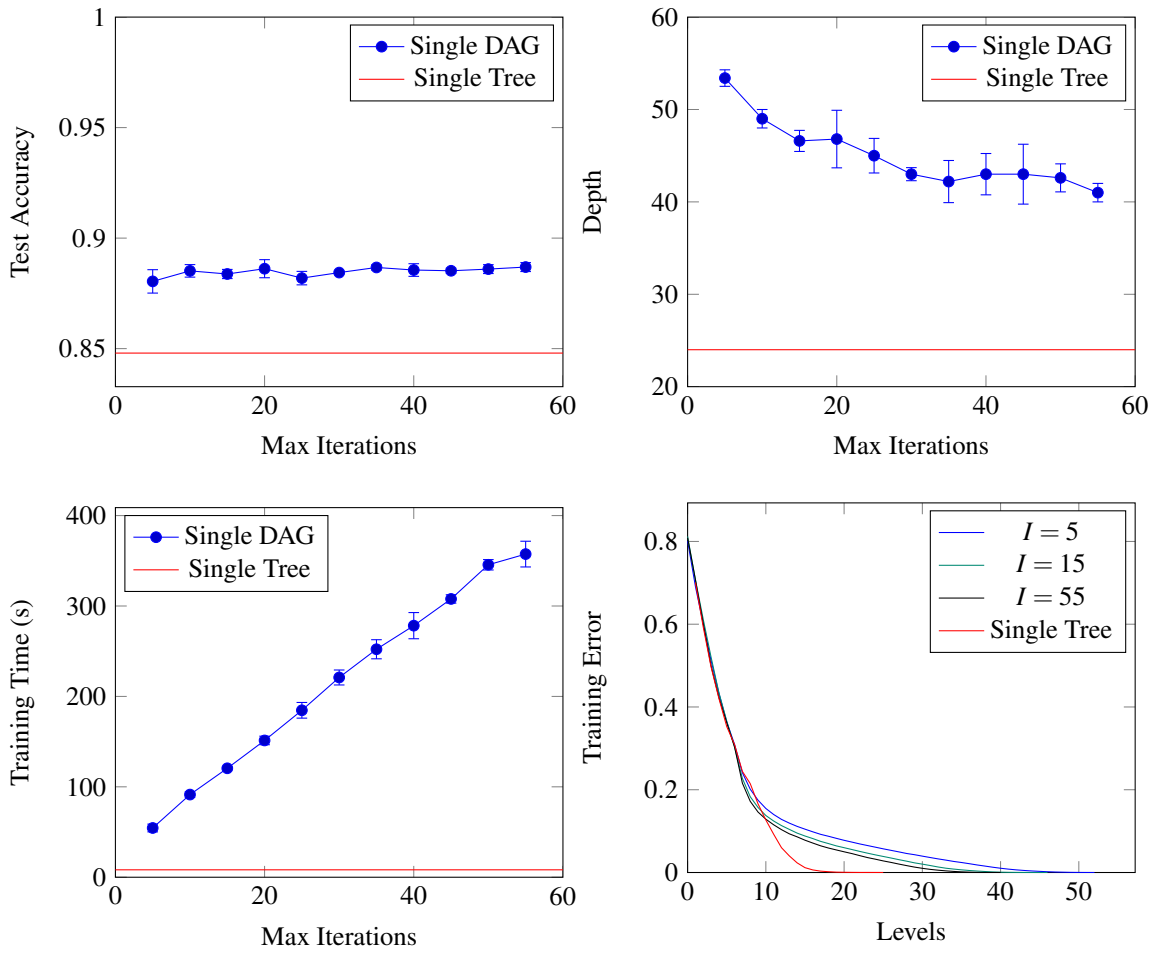


Figure 7: Figures show the influence of the maximum number of iterations on various output measurements.

5.2 Experiment 2: Performance of Ensembles

In this experiment, we investigate the performance of decision jungles. For this purpose, we trained ensembles of DAGs of various depths with and without bagging. When trained with bagging, each DAG/tree was trained using 60,000 randomly chosen (with replacement) training examples. All DAGs are trained with a maximum number of 35 iterations and each experiment has been conducted five times. Figure 8 reports the average values.

There are two notable observations. First, the significant gain in accuracy that was observed in the first experiment is no longer evident. The two kinds of ensembles yield approximately equal test accuracies; tough decision jungles still perform slightly better. Second, in stark contrast to random forests, bagging does not improve the resulting test accuracy of decision jungles. Altough one would expect the ensemble to perform better when the individual classifiers are less correlated, the exact opposite is the case.

5.3 Experiment 3: Child Node Initialization

Shotton et al. report that initializing the child nodes as discussed in section 3.1.3 speeds up convergence. In this experiment, we trained 10 DAGs with random initialization and 10 DAGs with the proposed initialization. Figure 9 reports the convergence speed (measured by the training error) and the test accuracy.

The experiment shows that the proposed initialization indeed increases convergence speed slightly. As a side effect, the initialization also slightly improves the test accuracy. Because it does not add a significant computational overhead, this initialization method if preferable over random initialization.

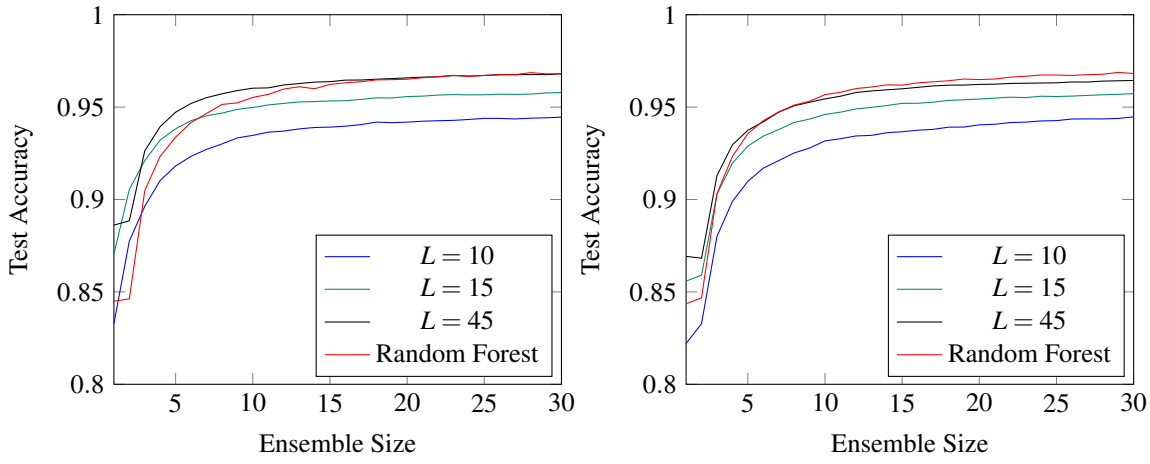
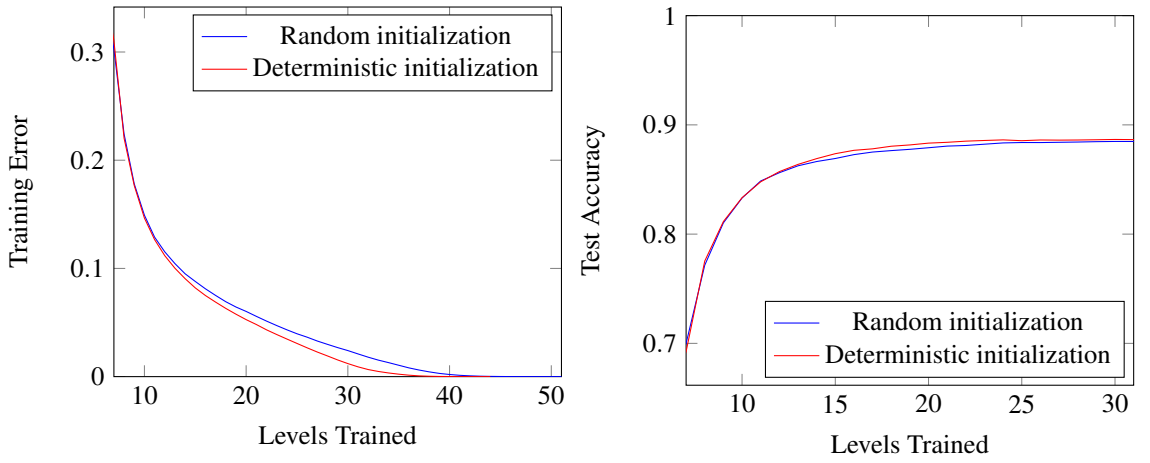


Figure 8: Figures show the test accuracy after n DAGs/trees have been added to the ensemble.



(a) Convergence speed

(b) Generalization performance

Figure 9: Performance of the proposed initialization method and random initialization.

5.4 Experiment 4: Performance on various Data Sets

In this experiment, we compare the test accuracy of decision jungles to the test accuracy of random forests. For this purpose, in addition to the MNIST data set, we chose several data sets from the UCI Machine Learning Repository [BL13]. Whenever a data set did not have a dedicated test set, we performed a random five-fold cross validation[11]. For data sets that comprise categorical data, we used the feature map introduced in section 2.1. Table 2 lists all the data sets we used in this experiment. Each experiment has been conducted five times. Table 3 reports the average test accuracies for ensembles of 8 and 15 classifiers respectively.

In order to report implementation-independent results for the memory consumption, we estimate the model sizes based on the number of nodes. Table 1 reports the weighting factors we used for this estimation. The resulting memory consumptions of both approaches are reported in table 4 and figure 10

Regarding test accuracy, the most notable oberservation is that decision jungles perform consistently better then random forests. However, at 0.28% for ensembles of 8 classifiers and an even lower 0.166% for ensembles of size 15, the gain in accuracy is very small.

In contrast to the test accuracy, the observed reduction in memory consumption is significant. Especially for

[11] We created five different training and test sets by dividing a randomly shuffled set 4:1 into two sets.

high-dimensional data sets that comprise many training examples, decision jungles consume considerably less memory than random forests. With the exception of the shuttle data set, the average reduction in memory consumption if about 50%.

Node type	**Size**	**Attributes**
Internal (Tree)	16 byte	d_v 4 byte, θ_v 8 byte (double precision), l_v 4 byte
Internal (DAG)	20 byte	d_v 4 byte, θ_v 8 byte (double precision), l_v 4 byte, r_v 4 byte
Leaf (Tree/DAG)	$C \cdot 4$ byte	$h_v : \{1,...,C\} \mapsto \mathbb{R}\, C \cdot 4$ byte (single precision)

Table 1: The table lists the estimated memory consumption of individual nodes.

Data set	**Size**	**Features**	**Attributes**	**Ensemble size**	**Comment**
MNIST	60,000/10,000	784	numerical	8/15	
USPS	3,823/1,797	64	numerical	8/15	
CONNECT 4	67,557/-	42(126)	categorical	8/15	No test set
LETTER RECOGNITION	20,000/-	16	numerical	8/15	No test set
SHUTTLE	43,500/14,500	9	numerical	8/15	

Table 2: The table lists all the data sets with their attributes that have been used for this experiment. The feature dimensions reported in parentheses correspond to the feature dimensions after applying the feature map for categorical data.

	Decision jungles				**Random forests**			
	8 DAGs		15 DAGs		8 Trees		15 Trees	
Data set	Mean	Stdev.	Mean	Stdev.	Mean	Stdev.	Mean	Stdev.
MNIST	**95.72%**	0.13%	**96.38%**	0.09%	95.14%	0.20%	96.23%	0.16%
USPS	**94.65%**	0.5%	**95.95%**	0.2%	94.44%	0.30%	95.93%	0.52%
CONNECT 4	**81.17%**	0.22%	**81.98%**	0.15%	80.99%	0.46%	81.47%	0.66%
LETTER RECOG.	**94.73%**	0.57%	**95.73%**	0.55%	94.29%	0.43%	95.58%	0.48%
SHUTTLE	99.98%	0.01%	**99.99%**	0.01%	**99.99%**	0.00%	**99.99%**	0.01%

Table 3: The table lists the means and standard deviations of the test accuracies.

	Decision jungles (15 DAGs)			**Random forests (15 Trees)**		
Data set	#internal	#leaf	Size (*byte*)	#internal	#leaf	Size (*byte*)
MNIST	134,936	1,877.2	**2,773,808**	95,940.4	95,955.4	5,373,262.4
USPS	13,113.4	1,012	**302,748**	6,433.2	6,448.2	360,859.2
CONNECT 4	74,633	1,920	**1,515,700**	169,604.4	169,619.4	4,749,103.2
LETTER RECOG.	69,485.6	1,075.2	**1,501,532.8**	40,776.2	40,791.2	4,894,704
SHUTTLE	4,896.4	802.6	113,980	804.2	819.2	**29,251.2**

Table 4: Average number of nodes and the resulting average memory consumptions.

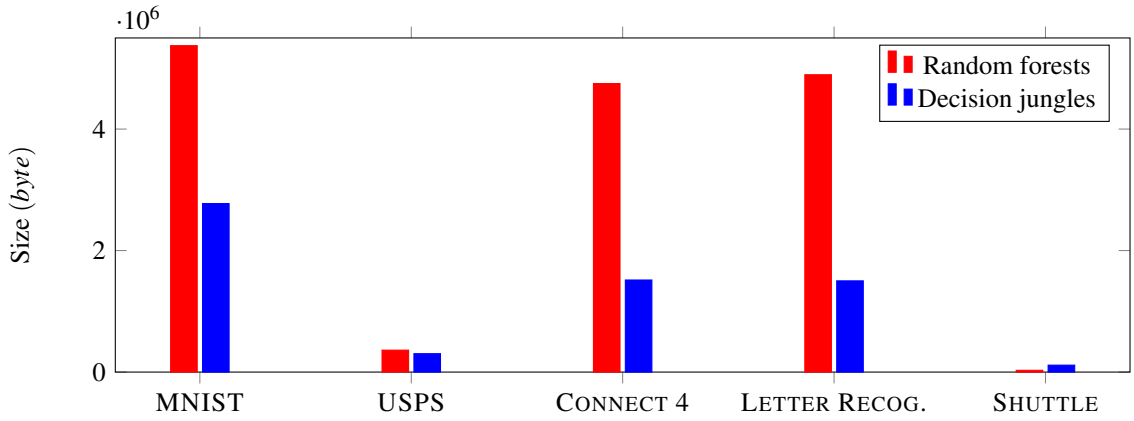


Figure 10: Comparison of the average memory consumption of random forests and decision jungles.

6 Conclusion

In this paper, we discussed the concept of decision jungles, which are ensembles of randomly trained decision DAGs (directed acyclic graphs) as proposed by Shotton et al. in [SSK+13a]. We presented the idea in a precise formal framework and highlighted the differences between binary decision trees and decision DAGs. We then presented the LSEARCH optimization algorithm that is used in order to train random decision DAGs level-by-level. After presenting the algorithm in its original form, we theoretically derived an efficient implementation. In the final chapter we evaluate the performance of decision jungles for the task of multi-class classification. For this purpose, we used our own implementation of decision jungles [Poh] and Piotr Dollár's implementation of random forests [Dol] in order to compare their performance on various data sets from the UCI Machine Learning Repository [BL13].

The experiments show that not only consume single decision DAGs significantly less memory than single binary decision trees, but they also increase test accuracy by a large margin. Although decision jungles still perform consistently better than random forests, the margin is no longer significant. At an average gain in test accuracy of 0.28%, decision jungles and random forests yield approximately the same performance.

In terms of memory consumption, decision jungles greatly outperform random forests. The experiments show an average reduction of approximately 50%. Hence, when the test accuracy is measured as a function of memory consumption, decision jungles show the significant gain in accuracy that is apparent in the paper of Shotton et al.

In our experiments, we solely focused on classification. Shotton et al. also report that decision jungles should be favored over random forests for the task of pixel-wise semantic segmentation of images. We did not verify this claim, which based on our observations seems likely. However, results that only focus on the absolute segmentation accuracy have still to be obtained.

In conclusion, decision jungles are a considerable alternative to random forests.

References

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[Dol] Piotr Dollár. Piotr's Image and Video Matlab Toolbox (PMT). `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`.

[Fis36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[HFL14] Alexander Hermans, Georgios Floros, and Bastian Leibe. Dense 3D Semantic Mapping of Indoor Scenes from RGB-D Images. In *International Conference on Robotics and Automation*, 2014.

[HMS66] Earl B Hunt, Janet Marin, and Philip J Stone. Experiments in induction. 1966.

[Poh] Tobias Pohlen. LibJungle - Decision Jungle Library. `https://bitbucket.org/geekStack/libjungle`.

[SGF+12] Jamie Shotton, Ross Girshick, Andrew Fitzgibbon, Toby Sharp, Mat Cook, Mark Finocchio, Richard Moore, Pushmeet Kohli, Antonio Criminisi, Alex Kipman, and Andrew Blake. Efficient human pose estimation from single depth images. *Trans. PAMI*, 2012.

[SSK+13a] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 234–242. Curran Associates, Inc., 2013.

[SSK+13b] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. supplementary material. http://research.microsoft.com/pubs/205439/DecisionJunglesNIPS2013Supplementary.pdf, 2013.

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

Appendices

A Definitions

Definition A.1 (in-/out degree) *Let $G = (V,E)$ be a graph. The* in-degree *$in(v)$ of a node $v \in V$ is the number of incoming edges.*

$$in(v) = |\{(w,v) \in E : w \in V\}| \tag{38}$$

Analog, the out-degree *$out(v)$ of a node $v \in V$ is the number of outgoing edges.*

$$out(v) = |\{(v,w) \in E : w \in V\}| \tag{39}$$

Definition A.2 (Rooted directed acyclic graph) *A* rooted directed acyclic graph (rooted DAG) *is a directed graph $G = (V,E)$ with the following properties*

- *There exists a unique* root node *$r \in V$ that has in-degree 0 and out-degree 2*
- *There exists a directed path from the root node to every other node*
- *G has no directed cycles*
- *Every other node $v \in V \setminus \{r\}$ either has $in(v) \geq 1$ and $out(v) = 2$ or it has $in(v) \geq 1$ and $out(v) = 0$ The former is called* internal node *and the latter is called* leaf node