Filter Forests for Learning Data-Dependent Convolutional Kernels

Sean Ryan Fanello[1,2] Cem Keskin[1] Pushmeet Kohli[1] Shahram Izadi[1]
Jamie Shotton[1] Antonio Criminisi[1] Ugo Pattacini[2] Tim Paek[1]

Microsoft Research[1] iCub Facility - Istituto Italiano di Tecnologia[2]

Abstract

We propose 'filter forests' (FF), an efficient new discriminative approach for predicting continuous variables given a signal and its context. FF can be used for general signal restoration tasks that can be tackled via convolutional filtering, where it attempts to learn the optimal filtering kernels to be applied to each data point. The model can learn both the size of the kernel and its values, conditioned on the observation and its spatial or temporal context. We show that FF compares favorably to both Markov random field based and recently proposed regression forest based approaches for labeling problems in terms of efficiency and accuracy. In particular, we demonstrate how FF can be used to learn optimal denoising filters for natural images as well as for other tasks such as depth image refinement, and 1D signal magnitude estimation. Numerous experiments and quantitative comparisons show that FFs achieve accuracy at par or superior to recent state of the art techniques, while being several orders of magnitude faster.

1. Introduction

Probabilistic models such as pairwise Markov random fields (MRF) and conditional random fields (CRFs) have been used for solving many pixel-wise labeling problems encountered in computer vision, including semantic segmentation, optical flow, image denoising, and stereo [3, 17]. These models allow the relationships between interacting variables (such as those corresponding to neighboring pixels) to be modeled easily, and typically lead to results which are smooth and respect edges in the image.

Despite impressive results, field-based models have two drawbacks: estimating the structure and parameters of models is hard, and inference of the *maximum a posteriori* solution under the model can be expensive. This has led researchers to investigate faster forest-based alternatives.

Decision forests [1, 4, 8, 23] have been used for problems such as body part classification [24] and organ detection [6]. While they enable efficient prediction, they also have a major drawback. In general, forest-based models make predictions under the assumption that output variables (such as pixel labels) are independent, and thus fail to enforce spatial smoothness. A recent exception is the work in [16] where the authors investigate the use of an iterative, stacking technique for the prediction of categorical (discrete) labels while respecting spatial context.

In this paper we propose *filter forests* (FF), a non-iterative forest-based predictor for the estimation of continuous variables, which can incorporate a learned model of spatial or temporal context. We apply FF here to the task of restoring corrupted n-dimensional signals.

Our main contribution is in the use of a highly-efficient multi-scale decision forest that is trained to discriminatively predict which filter should be applied at any given set of variables (*e.g.* pixels). FF has the following properties: (i) the forest recursively partitions the input signal such that a simple convolution kernel is appropriate at each leaf; (ii) we train the forest to minimize a new, regularized least squared error of the kernels at each leaf; (iii) we are able to achieve state of the art accuracy at a speed that is orders of magnitude faster than state of the art; and (iv) no field-based post-processing is required.

We evaluate FFs on a number of tasks including natural image denoising, depth image refinement, and 1D signal magnitude estimation. Experiments on a popular denoising image dataset show that our method achieves accuracy which is at par or superior to state of the art, but four orders of magnitude faster. Improved results are also demonstrated for depth image refinement and 1D signal magnitude estimation, hinting at the generality of our method. We believe FFs can be extended to many other related tasks such as learned edge detection, image sharpening, or even morphological filtering and discrete classification tasks.

1.1. Related Work

Convolution in Image Processing. Bilateral [27] and Wiener [30] are popular filters for noise removal in images. The former replaces the intensity at a pixel i with a weighted combination of nearby intensities, with weights depending on both intensity difference and spatial distance from i. The latter defines local filters conditioned on local variance estimates. Both filters try to preserve edges; but despite their popularity they do not always yield good denoising results.

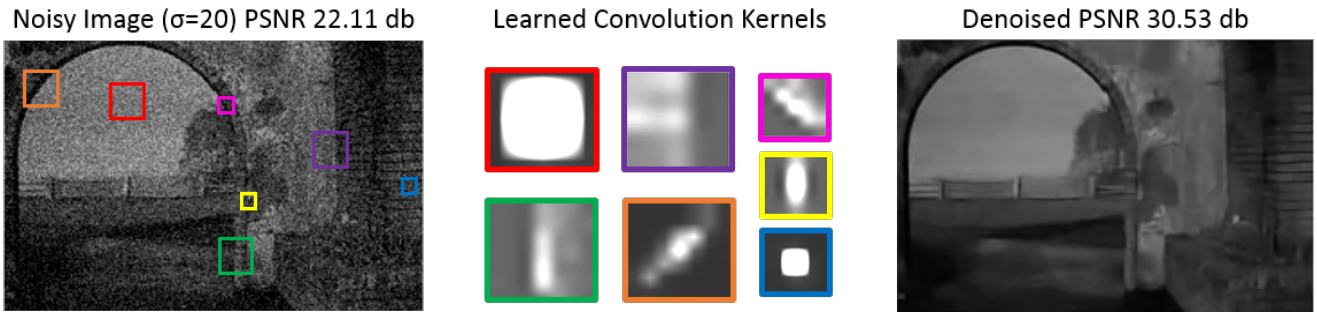

Figure 1. **Filter forests for learned, spatially-varying filtering**. Filter forests learn an optimal convolution kernel as a function of the local image appearance. **(left)** Input noisy image, with superimposed select locations. **(middle)** Learned convolution filters for corresponding locations (color-coded). **(right)** Denoised image. In textureless regions (red square) the filter forest learns to use large, isotropic smoothing filters. Instead, at object boundaries (orange or green squares) the forest learns to use directional, edge-preserving filters.

Further adaptive techniques include non-local means [5], K-SVD [11], LSSC [18] and CSR [10]. One of the most accurate methods, BM3D [9], uses local similarity of noisy patches to obtain a single prediction.

Filtering for Encouraging Spatial Coherence. A number of papers have considered the use of filtering for pixel labeling tasks. Convolving a unary likelihood with a filter can be used for inferring solutions to problems such as optimal flow, stereo and figure-ground segmentation [7, 20, 26]. Kontschieder *et al*. [16] integrated these ideas in a sequential classification approach where filtered outputs from one layer of predictors was given as input to the next layer of predictors. These methods are efficient, but their modeling power is limited by the fact that only a single fixed form of filter function is used. In contrast, our method uses a tree to select the filter to be applied from a potentially unboundedly large dictionary. Our method can be seen as a generalization of the convolution model, where the kernel is no longer fixed for the entire image, but varies spatially (or temporally), conditioned on the local appearance of the input signal.

Decision Trees for Structured-Output Prediction. The use of decision trees in combination with random fields for the purposes of image denoising has been investigated in [14, 19]. There, Nowozin *et al*. have shown how arbitrary, data-dependent pairwise potentials in CRF models can be encoded using a decision tree, and learned efficiently. Decision trees have also been used for reducing the computational complexity of inference methods. Shapovalov *et al*. [22] recently showed how the inference machines framework proposed in [21] can be used in conjunction with decision forests to efficiently assign class labels to 3D points. Our work differs from these methods in that it does not require explicit inference, it is non-iterative and involves a simple (yet spatially-varying) convolution operation.

2. Method

Many problems in image processing and computer vision can be formulated as a regression problem where we want to learn a mapping $f_{\mathbf{w}} : \mathcal{X} \to \mathcal{Y}$ from the space $\mathcal{X}$ of inputs to the space $\mathcal{Y}$ of outputs that is parameterized by some parameters $\mathbf{w}$. Learning these predictors is generally formulated using the principle of empirical risk minimization:

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} \sum_{i \in T} L(y_i^{\mathrm{GT}}, f_{\mathbf{w}}(x_i)) \qquad (1)$$

where T is the training data composed of pairs of input and expected (ground truth) output pairs (x_i, y_i^{GT}), and the function L computes the loss of predicting $f_{\mathbf{w}}(x_i)$ when the true solution is y_i^{GT}.

As an example, consider signal processing tasks that require recovering the original signal y, or some of its features (like edges), from possibly noisy observations x. The type of the noise can be additive, such that $x = y + \epsilon$, or multiplicative, Poisson, *etc*. Convolution of x with a kernel $\mathbf{w}$ is considered to be an efficient method to obtain an approximation to the original signal y. If we assume that the form of the prediction function $f_{\mathbf{w}}(x_i)$ is a convolution and the loss is the absolute value, then we can find the optimal kernel as:

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} \sum_{i \in T} \|y_i^{\mathrm{GT}} - \mathbf{x}_i^\top \mathbf{w}\|^2 \qquad (2)$$

where i is a data point index, and $\mathbf{x}_i$ is the k dimensional context of data point i. However, application of a fixed kernel homogeneously to the entire signal rarely gives satisfactory results, because signal and noise characteristics may vary throughout the signal. Our aim is to train a model that efficiently matches a data point and its context $\mathbf{x}_i$ with the optimal kernel $\mathbf{w}$ learned from a training set.

We propose solving this problem with random forests that store optimal filters $\mathbf{w}$ at the leaves and learn to efficiently assign these to the input $\mathbf{x}_i$. Notably, these filters can be local, specific and simple, yet the forest can still handle complicated types of noise (like a piecewise linear function converging to a continuous function). For instance, FF can handle multiplicative noise by approximating its response with additive noise at each leaf. Therefore, we choose to store linear filters that are learned by assuming additive noise, for which there is an effective closed form solution via least squares optimization [12]:

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{y} - \mathtt{X}\mathbf{w}\|^2 \,, \tag{3}$$

where $\mathbf{y} \in \mathbb{R}^N$ is the vector of the N ground truth (noise free) values, and $\mathtt{X} \in \mathbb{R}^{N \times k}$ is the matrix of N observations where each row is an $\mathbf{x}_i^\top$.

Note that the linear filtering $\mathtt{X}\mathbf{w}$ could be replaced with a non-linear function $f_{\mathbf{w}}(\mathtt{X})$ suitable for other types of problems. Examples include a median filter for different type of noise, or a sigmoid function that could cast a discrete labeling problem into a continuous one, so that it can be solved via regression.

We will now describe this model in more detail. From this point on we will focus on a specific application, namely image denoising, to make the explanations more intuitive.

2.1. Filter Forests

We use regression forests to learn a partitioning of the space of image patches $\mathbf{x}$ such that within each partition a simple linear convolution produces (measurably) good results. We will show: i) how to train a FF to minimize a well-defined denoising loss from labeled training data, ii) how our regularizer encourages edge-preserving filters, and iii) how to learn an appropriate scale for each kernel. Our experiments in Sec. 3 show that at test time the FF can produce accurate denoising at a very high speed. Below, we give a brief review of standard decision forests, before presenting details on FFs.

2.1.1 Conventional decision forests

A forest is an ensemble of T decision trees [1, 4, 8]. Each tree consists of non-terminal (split) and terminal (leaf) nodes. At test time a pixel i is passed into the root node. At each split node j, a split function $f(i; \boldsymbol{\theta}_j)$ is evaluated. This computes a binary decision based on some function of the image that surrounds the pixel i, based on learned parameters $\boldsymbol{\theta}_j$.[1] Depending on this binary decision, the pixel passes either to the left or right child. When a leaf is reached, a stored histogram over class labels (for classification) or a density over continuous variables (for regression) is output.

Each tree in the forest is trained independently. A set of example pixel indices i and their corresponding ground truth labels are provided. Starting at the root, a set of candidate binary split function parameters $\boldsymbol{\theta}$ are proposed at random, and for each candidate, the set of training pixels is partitioned into left and right child sets. An objective function (typically information gain for classification problems) is evaluated given each of these candidate partitions, and the best $\boldsymbol{\theta}$ is chosen. Training then continues greedily down the tree, recursively partitioning the original set into successively smaller subsets. Training stops when a node reaches a maximum depth D or contains too few examples.

[1] Function f can be any arbitrary function of the image region surrounding pixel i and is not limited to using just the patch $\mathbf{x}_i$.

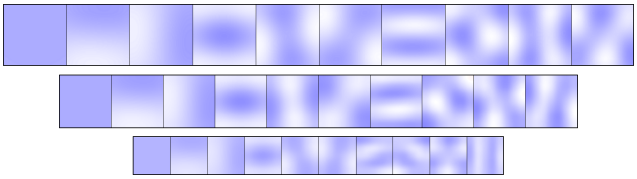

Figure 2. **PCA-based image features.** The first 10 principal components (shown in false color), computed from noisy training patches at scales 11x11, 7x7, and 3x3. They represent smooth regions, edges, corners as well as more complex patterns. Note that these PCA components are only used to compute the features used by the decision forest to partition the appearance space; the final filtered output image is computed directly from the noisy input.

2.1.2 Details on filter forests

Training data. For FFs, we use pairs of noisy image patches $\mathbf{x}_i$ and their corresponding noise-free ground truth values y_i (for the central pixel only).[2] As is standard with denoising techniques, for our experiments we generate training data by adding synthetic noise of particular characteristics (*e.g.* Gaussian with $\sigma = 20$ grey levels) to the noise-free ground truth images.

Features types. We employ two types of image features in FFs. The notation $\boldsymbol{\theta}$ is used to encapsulate which type of feature is used, the parameters of that feature, and a threshold applied to a scalar value that results in the required binary decision. All aspects of $\boldsymbol{\theta}$ are learned at each split node through randomized optimization [8].

The first type of feature is a multi-scale filter bank (as illustrated in Fig. 2). At each of the resolutions 3x3, 5x5, 7x7, 9x9, and 11x11, a PCA is performed on all the noisy training patches. For both training and testing, the filter bank is applied at each resolution as a convolution across the image. We allow two sub-types of this feature: unary and pairwise. The unary feature simply takes the response at pixel i of one of the top 10 PCA components in one of the resolutions, whereas the pairwise feature takes the difference of responses (both at pixel i) between two different PCA components in one of the resolutions. The split function f then simply applies the learned threshold to the resulting scalar value. The choice of unary/pairwise, the PCA components, the resolutions, and the best threshold are all learned through randomized node optimization. We use PCA eigenvectors instead of predefined filter banks [14] because of PCA's generality across different types of signal. Moreover, PCA ensures that the principal directions represent suitable features for axis-aligned splits at internal nodes.

The second type of feature is based on a local estimate of uniformity at each pixel. At each scale of 3x3, 5x5, 7x7, 9x9, and 11x11, we compute the variance of each image

[2] While we use $\mathbf{x}_i$ to conveniently denote a particular patch of the noisy image, for neighboring pixels these patches will share values, and the individual $\mathbf{x}_i$s need not be explicitly computed.

patch. We only employ a unary variant of this type of feature. The function f applies the learned threshold to the variance response at pixel i at one of the scales.

Training objective. One of the main contributions of our work is the form of the training objective. Let $\mathcal{S}_j$ denote the subset of training data that arrives at any node j of the tree. The split function at node j is parameterized with parameters θ and splits $\mathcal{S}_j$ into the left and right children subsets $\mathcal{S}_j^{\mathrm{L}}(\theta)$ and $\mathcal{S}_j^{\mathrm{R}}(\theta)$ respectively. The training algorithm selects the parameters of the split function by minimizing the energy $E(\boldsymbol{\theta}, \mathcal{S}_j)$ as

$$\boldsymbol{\theta}_j = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, E(\boldsymbol{\theta}, \mathcal{S}_j)\ , \tag{4}$$

with E is defined as the weighted sum of energies of the two child nodes

$$E(\boldsymbol{\theta}, \mathcal{S}_j) = \sum_{c \in \{\mathrm{L},\mathrm{R}\}} |\mathcal{S}_j^c(\boldsymbol{\theta})| E^c(\mathcal{S}_j^c(\boldsymbol{\theta}))\ . \tag{5}$$

The energy of each child node is computed as

$$E^c(\mathcal{S}_j^c) = \|\mathbf{y}_j^c - \mathtt{X}_j^c \mathbf{w}^\star\|^2\ , \tag{6}$$

where

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \left(\|\mathbf{y}_j^c - \mathtt{X}_j^c \mathbf{w}\|^2 + \|\Gamma(\mathtt{X}_j^c, \mathbf{y}_j^c)\mathbf{w}\|^2 \right)\ , \tag{7}$$

Here, $\mathtt{X}_j^c$ and $\mathbf{y}_j^c$ are computed from $\mathcal{S}_j^{\mathrm{L}}$ and $\mathcal{S}_j^{\mathrm{R}}$, and represent the set of noisy training data patches and the set of noise-free ground truth values that have reached the left ($c = \mathrm{L}$) or right ($c = \mathrm{R}$) child respectively of node j. Further, $\Gamma(\mathtt{X}, \mathbf{y})$ represents a *data-dependent* regularization weighting matrix.

Data-dependent regularized training. The above mentioned optimization task would be a standard regularized least squares problem if not for the use of $\Gamma(\mathtt{X}, \mathbf{y})$. As will become clearer later, Γ encourages edge-preserving regularization. The closed form solution of the minimization problem (7) can be computed as

$$\mathbf{w}^\star = (\mathtt{X}^{\mathrm{T}}\mathtt{X} + \Gamma^{\mathrm{T}}\Gamma)^{-1}\mathtt{X}^{\mathrm{T}}\mathbf{y}\ . \tag{8}$$

If Γ were set to identity, then the regularization term in (7) would encourage kernels to have smaller norms. In this work, we instead investigate the use of a data-dependent regularization, where smaller entries in $\mathbf{w}$ are encouraged *only when they differ from the pixel i at the center of the patch*.

To achieve this, we use a diagonal Γ matrix of size $p^2 \times p^2$ computed as a function of the data matrix $\mathtt{X}$ and ground truth values as follows:

$$\gamma_d(\mathtt{X}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (x_{i,d} - y_i)^2\ , \tag{9}$$

where $d \in \{1, \ldots, p^2\}$ indexes pixels within the patch, N is the total number of samples (*i.e.* the number of rows) in $\mathtt{X}$,

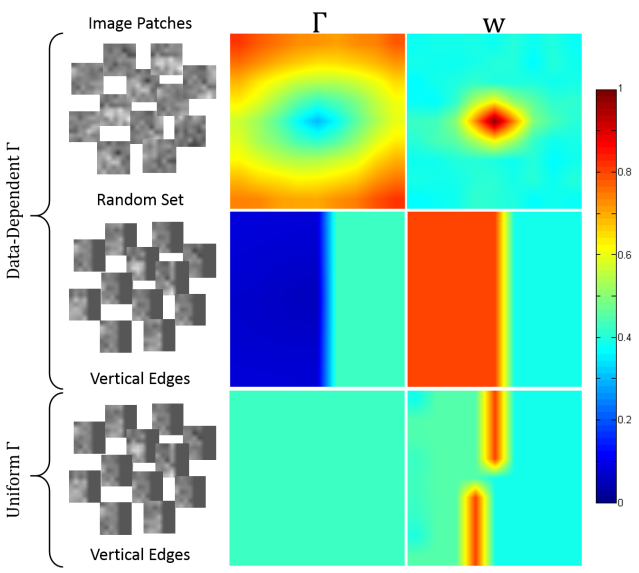

Figure 3. **The importance of the Γ weighting matrix**. **Column 1:** different subsets of image patches that comprise $\mathtt{X}$. **Column 2:** the resulting $\Gamma(\mathtt{X}, \mathbf{y})$ weighting matrices. **Column 3:** the learned filters $\mathbf{w}^\star$. Rows 1 and 2 use the Γ computed in (9), while row 3 uses $\Gamma = \mathtt{I}$. Row 1 uses a randomly chosen set of patches, while rows 2 and 3 use the same set of patches with a vertical edge just to the right of the central pixel. Our weighted regularization encourages edge-aware filters. See text for more details.

and $x_{i,d}$ and y_i are entries in $\mathtt{X}$ and $\mathbf{y}$ respectively. Intuitively, the Γ matrix indicates which regions of the noisy image patch are most likely to describe the ground-truth value y_i at the center of the patch. A pixel d with a larger value of γ_d will discourage the filter $\mathbf{w}$ from using pixel d, while smaller values of γ_d will encourage the use of pixel d.

To illustrate the importance of this regularization term, we consider two sets of patches that are shown in Figure 3: randomly generated (first row) and with vertical edges (second and third rows). From these sets we compute Γ (second column) and hence using (8) the filter $\mathbf{w}$ (third column). Using random patches we obtain a filter that is, as we might expect, somewhat isotropic. In the second row we select patches containing vertical edges where the central pixel is to the left of the edge. As expected, the regularizer encourages the filter to consider only the pixels to the left of the edge that are likely to be good predictors of the central pixel. This is an example of how the regularization encourages edge-preserving filters.

The third row in Figure 3 shows the solution obtained for patches containing vertical edges by setting $\Gamma = \mathtt{I}$. In this final case all dimensions d of the filter are equally regularized (no data dependence), and the result is a poor filter that is unlikely to generalize well.

Multi-scale filter learning. The optimization of (7) is obtained by testing different filter sizes, from 3×3 to 11×11, and selecting the one corresponding to the lowest error. This simple scheme allows us to compute the appropriate kernel size for each leaf, depending on the surrounding context of the pixel. Example of such data-dependent kernels are shown in Figure 1. Smooth areas are likely to be restored by large kernels (in red), whereas fine details are recovered using smaller and/or directional filters (orange).

The training phase ends when the tree reaches a certain depth or the number of examples is too small. Each leaf stores a different filter kernel $\mathbf{w}_j^\star$, learned using the regularized least squares error minimization computed over the subset of labeled training examples that reached leaf j (8).

Test time signal filtering. At test time, for each tree, the test pixel i is sent into the root node. Then each split node sends it to one of its child nodes until it reaches a leaf $j(i)$. Thanks to the nodes hierarchical structure, such descent is very efficient. The kernel $\mathbf{w}_{j(i)}$ stored at leaf $j(i)$ is looked up, and applied as a dot product to the noisy image patch at i, to produce the denoised output $\hat{y}_i = \mathbf{x}_i^\top \mathbf{w}_{j(i)}$. Thus, a different kernel is applied to each input patch, conditioned on the patch appearance. In a forest, each tree is applied in turn and the results averaged. Different trees can be handled independently and in parallel for further efficiency.

Algorithm Complexity. The runtime complexity of the algorithm depends linearly on the size of the signal, *i.e.* $\mathrm{O}(|\mathtt{X}|)$. In particular, for image denoising, the number of operation per pixel is:

$$10O(p^2) + T(\mathrm{O}(p^2) + \mathrm{O}(d)) \tag{10}$$

where p^2 is the biggest patch size (in our case 11×11). The first term $10\mathrm{O}(p^2)$ is due to the PCA projector computations, $T\mathrm{O}(p^2)$ is the dot product between the predicted filters $\mathtt{W}$ and the patch. Finally $T\mathrm{O}(d)$ is the cost for descending the forest, with T trees of depth d. This cost being negligible, we can approximate the whole running time to $\sim (T+10)\mathrm{O}(p^2)$ per pixel. These operations can be parallelized since the final result does not depend on the results of the other image pixels. This further decreases the overall running cost. Notably, the cost per pixel is much lower than one of the fastest state of the art methods [9] (see time comparisons in Table 1).

Implementation details. During training, for each tree node, we use reservoir sampling [28] to select a subset of example pixels i. This speeds up the training process while ensuring a certain degree of randomness. Only the variance feature at multiple scale is allowed to be selected in the very first levels of the tree. This helps separate smooth regions from textured ones early on.

3. Results and Comparisons

This section presents results for FFs applied to each of the following tasks: (i) image denoising, (ii) depth image refinement, and (iii) 1D dynamical system filtering. For the first problem, we performed exhaustive evaluations using the popular BSDS500 benchmark [2], comparing our method with current state of the art algorithms. The depth denoising task has been evaluated using the 7 Scenes dataset from [25]. For the third experiment we designed a real, noisy dynamical system, and compared the predictions of filter forests with those from the standard Kalman and Least Mean Squares (LMS) filters. These experiments demonstrate the flexibility and effectiveness of the proposed framework in different application domains.

3.1. Image Denoising

For image denoising experiments, we use the same protocol as in [14]; images are re-scaled by a factor of 0.5, the validation set is used for parameter tuning, and the final model is evaluated on the test set. To form training images, Gaussian noise with zero mean and standard deviation $\sigma \in \{20, 30, 40, 50\}$ is added to every pixel in the images independently. We trained our method on 300 images from training and validation set, and the test set is composed of 200 images. The results are shown in Table 1. The proposed FF-based method gets competitive PSNR scores with an average running cost of 0.025 seconds per image. To improve the accuracy further, we apply the collaborative Wiener filter as proposed in [9] in a post-processing step ($\mathrm{FF}_{\mathrm{Wiener}}$ in the table). This drastically enhances the accuracy of the method for a small cost in speed, making it better than all the other methods except for $\mathrm{RTF}_{\mathrm{ALL}}$[14], which uses denoised images computed with four state-of-the-art methods as its input. For input noise $\sigma = 20$ we match the accuracy of this algorithm, with a running time of 0.125 seconds compared to 1275 seconds required by $\mathrm{RTF}_{\mathrm{ALL}}$. Hence, we conclude that, for more realistic levels of noise, FF performs as well as the state-of-the-art, while being 4 orders of magnitude faster. Applying the same post-processing step to other methods did not yield significant increases in accuracy in our experiments. Qualitative comparisons are given in Fig. 4.

3.2. Depth Image Refinement

We conduct experiments on the first sequence of each scene in the 7 Scenes Dataset [25], which has a total of 6500 frames. Synthetic 'ground truth' depth was computed using a KinectFusion [13] reconstruction of the scene. Training images are generated by adding depth-dependent Poisson noise to each pixel. We train an FF (four trees of depth 12 with multi-scale patches) on 1000 frames selected from the Chess scene, and test the model on the other scenes. The quantitative results are reported in Table 2, showing that the proposed approach outperforms the compared methods. This

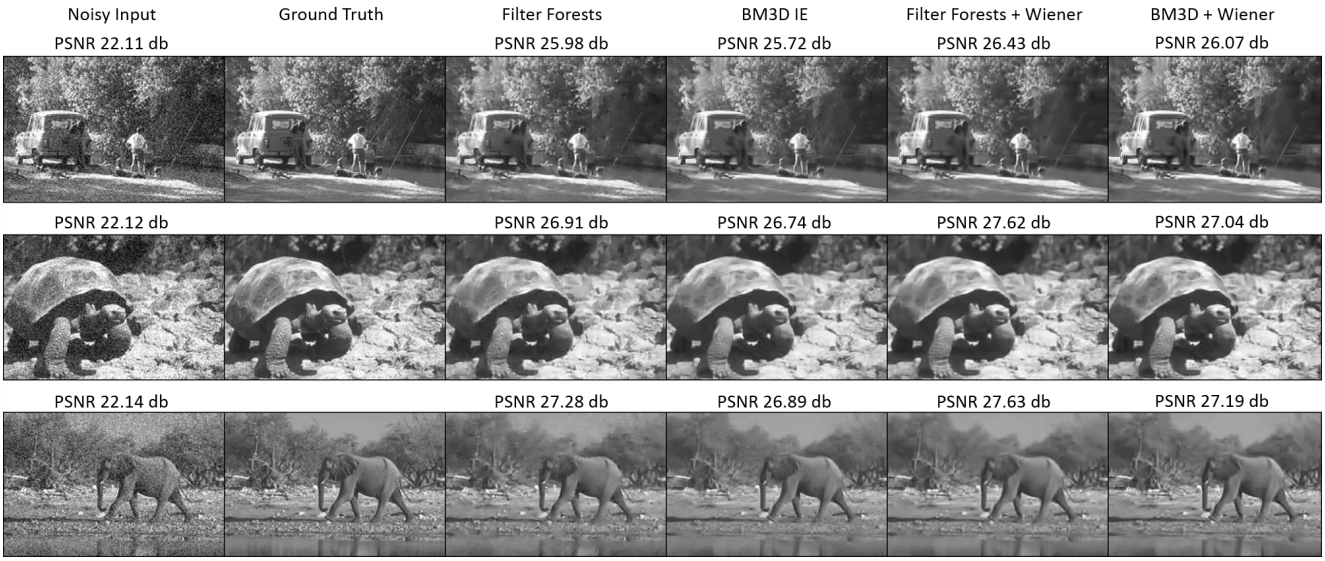

Figure 4. **Qualitative results of natural image denoising.** From left to right: noisy input, ground truth, FF output, BM3D first phase, FF with Wiener filtering, BM3D with Wiener filtering. Best viewed digitally at high zoom.

Method	$\sigma = 20$	$\sigma = 30$	$\sigma = 40$	$\sigma = 50$	sec.
Wiener	25.73	24.79	23.95	23.19	**0.001**
FF	28.75	26.55	25.32	24.51	**0.025**
FF$_{\text{Wiener}}$	**29.65**	**27.48**	**26.15**	**25.25**	**0.125**
RTF$_{\text{PLAIN}}$ [14]	28.95	26.97	25.71	24.76	0.7
BM3D [9]	29.25	27.32	25.98	25.09	0.9
EPLL [31]	29.38	27.44	26.17	25.22	38
CSR [10]	29.17	27.24	25.91	24.99	124
LSSC [18]	29.40	27.39	26.08	25.09	172
RTF$_{\text{ALL}}$ [14]	**29.67**	**27.72**	**26.43**	**25.51**	1275

Table 1. **Denoising results (PSNR) and comparisons on the BSDS500 benchmark.** Comparison of FF with state-of-the-art denoising algorithms. The last column reports average running times (in seconds) for a single image (241×161). All experiments run on a 4-core Intel Xeon machine (2.4GHz).

also demonstrates that FFs can handle noise types other than additive noise. In this particular application, collaborative Wiener filtering (FF$_{\text{W}}$) did not significantly improve the accuracy. Qualitative examples are shown in Fig. 5.

FF	FF$_{\text{W}}$	Wiener	Bilateral	LMS	BM3D [9]
35.61	35.63	32.29	30.95	24.37	35.46

Table 2. **Test set PSNR (db) for the depth refinement experiment.** We compare our method with other popular filters: Wiener, Bilateral, LMS, and BM3D [9]

3.3. 1D Dynamical Signal Filtering

Here, we demonstrate the application of FFs to a non-image modality. Consider a discrete dynamical system of the form:

$$\begin{aligned} \mathbf{x}_k &= \mathtt{A}_k \mathbf{x}_{k-1} + \mathtt{B}_k \mathbf{u}_k + \boldsymbol{\omega}_{k-1} \\ \mathbf{z}_k &= \mathtt{C}_k \mathbf{x}_k + \boldsymbol{\nu}_k \end{aligned} \tag{11}$$

where: $\mathbf{x}_k \in \mathbb{R}^n$ is the internal state at time k evolving from its previous state at time $k-1$; $\mathtt{A}_k \in \mathbb{R}^{n \times n}$ is the transition matrix; $\mathtt{B}_k \in \mathbb{R}^{n \times n}$ is the input matrix accounting for the contribution of the control vector $\mathbf{u}_k$; $\mathbf{z}_k \in \mathbb{R}^m$ is an observation of the current (noisy) state $\mathbf{x}_k$; $\mathtt{C}_k \in \mathbb{R}^{m \times n}$ is a transformation matrix that maps the state into the observations; and $\boldsymbol{\omega}_{k-1} \in \mathbb{R}^n$ and $\boldsymbol{\nu}_k \in \mathbb{R}^m$ are the zero mean process noise with covariance matrices $\mathtt{Q}_k \in \mathbb{R}^{n \times n}$ and $\mathtt{R}_k \in \mathbb{R}^{m \times m}$ respectively. In this setting we consider a time series of t observations $\mathbf{z}_k = \mathbf{z}_{k-t}, \ldots, \mathbf{z}_k$ with the aim of learning a filter $\mathtt{w}$ such that $\mathbf{x}_k = \mathbf{z}_k^\top \mathtt{w}$. The model used for image denoising can thus be applied to this scenario without any modification.

As a practical example of Eq. 11, we refer to the model of an electrical DC motor, which is described by the following state-space representation

$$\begin{aligned} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_k &= \begin{pmatrix} 0 & 1 \\ 0 & -a \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_{k-1} + \begin{pmatrix} 0 \\ b \end{pmatrix} V_k + \mathbf{w}_{k-1} \\ z_k &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}_k + \nu_k \end{aligned} \tag{12}$$

where the parameters $a = \frac{1}{\tau}$ and $b = \frac{K}{\tau}$ are known from the motor's datasheet (see [15] for details). Here x_1 and x_2 are the position [rad] and the velocity [rad/s] of the motor respectively, V_k is the motor armor voltage [Volt]. Given a series of noisy observations $z_1, \ldots, z_k$ from the load encoder, we want to predict the current state of the system x_k that is the denoised position of the motor encoder [rad]. For the first experiment, the signal is corrupted with a Gaussian noise (zero mean, variance $\sigma = 0.01$), and the proposed model is compared with a Kalman filter. In this setting, the Kalman filter is the optimal estimator if the noise is white and the covariances of the noise are known [29] (*i.e.* it minimizes

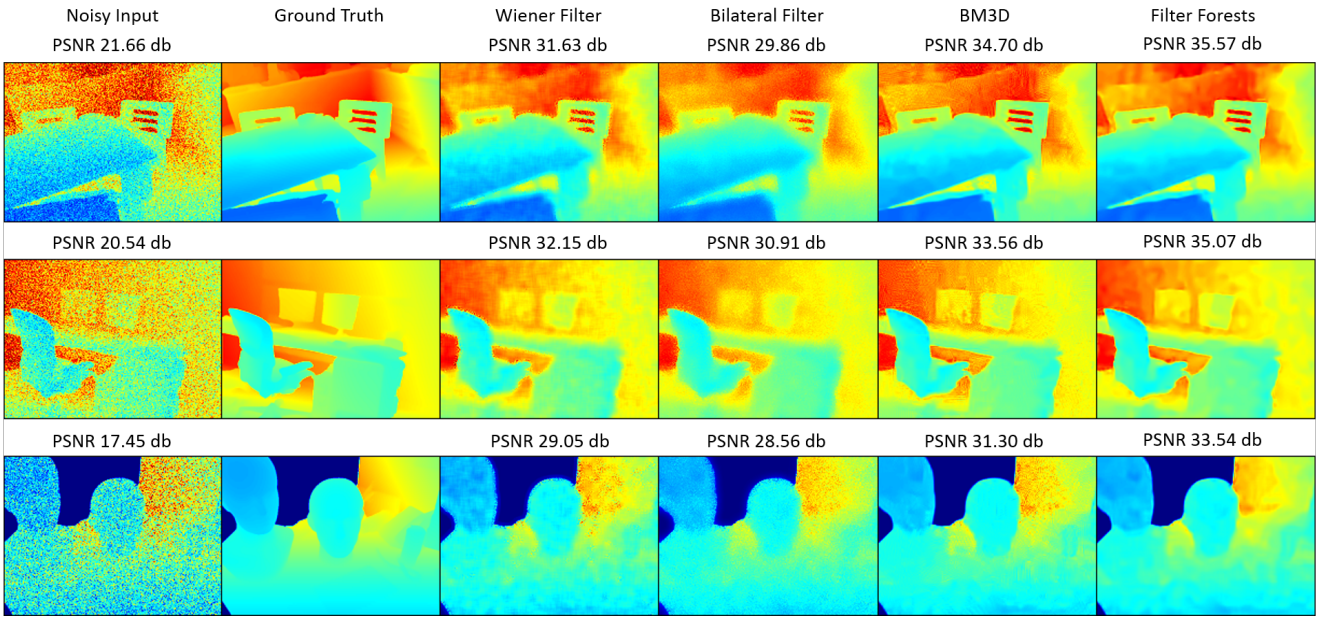

Figure 5. **Qualitative results of depth refinement.** From left to right: noisy input, BM3D, Wiener filter, bilateral filter, LMS filter, FF. FF produces visually better results.

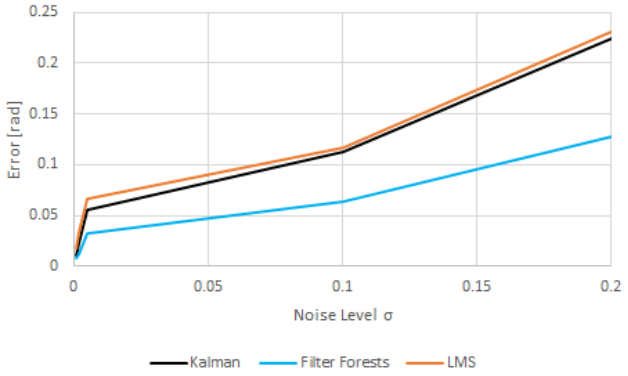

Figure 6. **Dynamical system filtering in presence of structured noise.** We show error as the noise level σ is varied. FFs systematically outperform LMS and the Kalman filter.

the mean square error of the estimated output).

We generate 100 sequences of 2000 samples for the training phase, and test on 100 new noisy sequences. For a noisy signal with 0.0097 rad MSE, the Kalman filter achieves an error of 0.0018 rad, and FFs manage to converge to the optimal filter with 0.00184 rad of error, without any assumption on the system or noise model. In Fig. 7, left plot, we show a denoised sequence. Notably, a least mean square filter `w` achieves an error of 0.0033 rad.

In a second experiment, we generate input dependent noise of the form $w_k = \nu x_k$, where ν is a random variable with zero mean and variances $\sigma \in \{0.01, 0.02, 0.05, 0.1, 0.2\}$. When the noise is structured, the Kalman filter is not guaranteed to be optimal. Here, FFs systematically outperform the Kalman and the LMS filters at every noise level. Quantitative results are reported in Fig. 6, and qualititative results in Fig. 7 (right).

4. Conclusion

We have proposed FF as an efficient, novel discriminative approach for signal filtering and restoration. FFs are a non-linear, data-adaptive extension of convolution-based filtering, but can be further extended to other filter types or non-linear functions. A new data-driven, regularized training objective allows us to adapt decision forests to the task of regressing continuous variables, while exploiting their spatio-temporal context.

Numerous experiments and quantitative comparisons show that FFs achieve accuracy, which is at par or better than recent state of the art algorithms; and orders of magnitude faster. We believe FFs can readily be extended to other applications such as edge detection, sharpening, inpainting, superresolution and many more.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7), 1997. 1, 3

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. PAMI*, 2011. 5

[3] A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011. 1

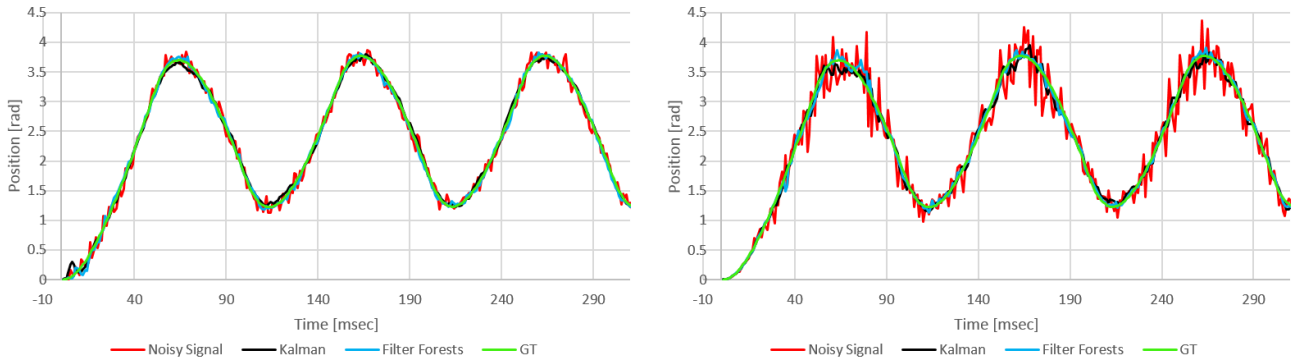

Figure 7. **Dynamical system filtering example.** Kalman filter (black line) and an FF(cyan line) are used to estimate the encoder position of a DC electrical motor (green line) from noisy observations (red line). Left: white Gaussian noise is added. In this setting the Kalman filter represents the optimal estimator, and FF learns a very close approximation. Right: input dependent multiplicative noise is generated. In this scenario our more flexible approach outperforms the Kalman filter.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1), 2001. 1, 3

[5] A. Buades, B. Coll, and J. M. Morel. A non-local algorithm for image denoising. In *Proc. CVPR*, 2005. 2

[6] A. Criminisi, D. Robertson, E. Konukoglu, J. Shotton, S. Pathak, S. White, and K. Siddiqui. Regression forests for efficient anatomy detection and localization in computed tomography scans. *Medical image analysis*, 2013. 1

[7] A. Criminisi, T. Sharp, C. Rother, and P. Perez. Geodesic image and video editing. *Proc. ACM SIGGRAPH*, 2011. 2

[8] A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013. 1, 3

[9] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Trans. Image Processing*, 2007. 2, 5, 6

[10] W. Dong, X. Li, L. Zhang, and G. Shi. Sparsity-based image denoising vis dictionary learning and structural clustering. In *Proc. CVPR*, 2011. 2, 6

[11] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. Image Processing*, 2006. 2

[12] S. Haykin and B. Widrow. *Least-Mean-Square Adaptive Filters*. Wiley, 2003. 2

[13] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. In *ACM UIST*, 2011. 5

[14] J. Jancsary, S. Nowozin, and C. Rother. Loss-specific training of non-parametric image restoration models: a new state of the art. In *Proc. ECCV*, 2012. 2, 3, 5, 6

[15] T. Kara and I. Eker. Nonlinear modeling and identification of a {DC} motor for bidirectional operation with real time experiments. *Energy Conversion and Management*, 2004. 6

[16] P. Kontschieder, P. Kohli, J. Shotton, and A. Criminisi. GeoF: Geodesic forests for learning coupled predictors. In *Proc. CVPR*, 2013. 1, 2

[17] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc.ICML*, 2001. 1

[18] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Proc. ICCV*, 2009. 2, 6

[19] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In *ICCV*, pages 1668–1675, 2011. 2

[20] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *CVPR*, pages 3017–3024, 2011. 2

[21] S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell. Learning message-passing inference machines for structured prediction. In *CVPR*, pages 2737–2744, 2011. 2

[22] R. Shapovalov, D. Vetrov, and P. Kohli. Spatial inference machines. In *CVPR*, pages 2985–2992, 2013. 2

[23] T. Sharp. Implementing decision trees and forests on a GPU. In *Proc. ECCV*. Springer, 2008. 1

[24] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient human pose estimation from single depth images. *IEEE Trans. PAMI*, 2013. 1

[25] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proc. CVPR*, 2013. 5

[26] M. W. Tao, J. Bai, P. Kohli, and S. Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. *Comput. Graph. Forum*, 31(2):345–353, 2012. 2

[27] C. Tomasi. Bilateral filtering for gray and color images. In *Proc. ICCV*, 1998. 1

[28] J. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 1985. 5

[29] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, 1995. 6

[30] N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. The MIT Press, 1964. 1

[31] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. In *Proc. ICCV*, 2011. 6