Michael Weißensteiner, BSc

Representation Learning with Random Forests for Image Classification

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Telematics

submitted to
Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr. Horst Bischof
Institute for Computer Graphics and Vision

Advisors

Dipl.-Ing. Dr. Samuel Schulter
NEC Laboratories America, Inc.

Dipl.-Ing. Georg Poier
Institute for Computer Graphics and Vision

Graz, Austria. November, 2016

To my beloved Son
Armin

"A picture is worth a thousand words."

Arthur Brisbane (1864 - 1936)

Abstract

Building an effective representation of images is one of the most critical parts in high-level computer vision applications like image classification or detection. The recent success of deep learning underpins the importance of a good image representation. Convolutional Neural Networks (CNNs) contribute to state-of-the-art image classification where multi-layer architectures allow to extract features on different levels of abstraction. However, such systems require tremendous amount of computational resources, which is an issue if working memory or processing speed is limited.

The works of Coates *et al.* [20, 21] show, that also efficient single-layer architectures enable high image classification performance, despite relying on unsupervised dictionary learning. However, there are two main issues: (i) this approach is especially sensitive to preprocessing parameters and (ii) still requires time-consuming convolutional feature extraction. To overcome these issues, we propose patch-based representation learning with Random Forests (RFs). This replaces the expensive convolutions by simple binary splitting-tests, providing a fast and direct way to extract image features. In line with this, we investigate two different representation learning approaches: Our first approach exploits the unsupervised dictionary learning method of Coates *et al.* and trains a RF on raw image patches. Experiments on MNIST-10 and CIFAR-10 show that the resulting image representations are able to improve classification performance. Moreover, one experiment, including training and testing, can be done within minutes on a multi-core CPU. Inspired by the success of our first approach, we then replace the first convolutional layer of a baseline CNN by a RF. Therefore we introduce an iterative end-to-end optimization for MNIST-10 classification, which enables an improvement of 17% over the baseline CNN.

Kurzfassung

Die effektive Erzeugung von Bildrepräsentationen ist einer der wichtigsten Arbeitsschritte von High-level Computer Vision Anwendungen, wie Bildklassifizierung oder Detektion. Jüngste Erfolge mit Deep-Learning unterstreichen die Bedeutung einer charakteristischen Bildrepräsentation. Convolutional Neural Networks (CNNs) sind Stand der Technik bezüglich Bildklassifizierung und ermöglichen durch einen mehrschichtigen Aufbau eine Extraktion von Bildmerkmalen, auch Features genannt, über verschiedene Stufen der Abstrahierung. Solche Netzwerke benötigen jedoch oft eine große Anzahl an Convolutions und sind daher mit einem sehr hohen Rechenaufwand verbunden, was zu Problemen führen kann, wenn Arbeitsspeicher oder Rechengeschwindigkeit begrenzt sind.

Die Arbeiten von Coates *u.a.* [20, 21] zeigen, dass auch effiziente einschichtige Architekturen hohe Bildklassifikationsraten ermöglichen, obwohl Unsupervised-Dictionary-Learning Methoden verwendet werden. Bei diesem Ansatz gibt es jedoch zwei Kernprobleme: (i) es besteht hohe Empfindlichkeit gegenüber Preprocessing-Parametern und (ii) es wird nach wie vor eine Faltungsextrahierung von Features durchgeführt. Mit dem Einsatz von Random Forests (RFs) lernen wir Patch-Repräsentationen und ersetzen somit die Faltungsextrahierung durch einfache binäre Entscheidungen. Dadurch erhalten wir eine effiziente und direkte Möglichkeit Features von rohen Bilddaten zu berechnen. In diesem Zusammenhang untersuchen wir zwei verschiedene Repräsentations-Lernmethoden: Unsere erste Methode nutzt den Unsupervised-Dictionary-Learning Ansatz von Coates *u.a.* und trainiert einen RF auf rohen Bild-Patch Daten. Experimente auf MNIST-10 und CIFAR-10 zeigen, dass dadurch erzeugte Bildrepräsentationen die Bildklassifizierung verbessern. Zusätzlich kann ein Experiment, das Training und Testen umfasst, innerhalb weniger Minuten auf einer Multi-Core CPU durchgeführt werden. Durch den Erfolg dieser ersten Methode inspiriert, ersetzen wir in weiterer Folge den ersten Convolutional-Layer eines Ausgangsnetzwerkes durch einen RF. Dabei erreichen wir, mittels iterativer end-to-end Optimierung, eine Steigerung von ca. 17% bei einer MNIST-10 Klassifikation.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First and foremost I would like to thank Samuel Schulter, who initiated this Master's Thesis during his PhD program at the Institute for Computer Graphics and Vision (ICG). He fairly supported me with his wide knowledge about Random Forests and Machine Learning where I particularly appreciate our variety of pleasant discussions that have always been inspiring and motivating. As time went on, Samuel finished his PhD and successfully applied for a job at NEC Laboratories America. Since Samuel left Austria and the ICG, I am deeply grateful that Georg Poier agreed to be my new supervisor who continued supporting me. I am especially thankful for his great advice and for helping me finishing this Thesis. Special thanks go to Horst Bischof as he provided an opportunity to research and author this Master's Thesis at the facilities of the ICG, which included a workplace and a desktop machine. Also I want to thank the ICG members and staff who all enabled a nice atmosphere as well as interesting conversations. Further I thank my colleagues for the great time and our enlightening coffee breaks.
Finally I am deeply grateful for my beloved Son Armin who was born during this Thesis, and for my beloved girlfriend Doris who greatly supported our new family.

Contents

List of Figures

List of Tables

1 Introduction

Computer vision tries to analyze or interpret visual information which requires extracting relevant image characteristics. The popular proverb "*A picture is worth a thousand words*" states that images may contain a huge amount of "words" or information which we also denote as image features. In computer vision, a process called *feature extraction* is responsible to extract relevant feature information and to produce an *image representation* that ideally encodes the visual "essence" of an image. Hence, building an effective representation of images is crucial for *image understanding*.

In the past, numerous works on image feature processing and encoding appeared [18, 21, 36, 68]. Examples are successful hand-crafted feature descriptors such as the keypoint-based SIFT-descriptor [58] or the HOG-descriptor [25] which encode image gradient information. Various kinds of such manually designed image features are commonly used and show good performance for specific tasks. Although extensive research has been dedicated to hand-crafted features, more sophisticated methods that use *learned* features or representations gained interest over the last few years [19, 21, 22, 57]. Recent works show that learned representations prove to outperform hand-crafted approaches for many tasks [4, 7, 17].

To learn image representations, machine learning approaches use unsupervised, supervised or semi-supervised methods. Since growing computational power allows to learn highly complex feature hierarchies, especially deep learning systems gained popularity. Nowadays, designing and employing hierarchical multi-layer architectures is an active research area where Deep Neural Networks (DNNs) [19, 64, 76] show state-of-the-art results for numerous computer vision applications.

Such multi-layer architectures usually use a hierarchy of convolutional layers, also called Convolutional Neural Nets (CNNs) [57]. Each layer learns the values of multiple convolution kernels that build feature maps or *feature channels* which are then processed by the next layer. Hence, with increasing network depth, higher levels of abstraction are learned which have shown to yield good generalization. This hierarchical concept has been

inspired by biological processes like those discovered by Hubel and Wiesel [46, 47] in the late 1950s. They investigated a cat brain and found two main types of cells within the visual cortex. The hierarchical arrangement from so-called "simple cells" up to "complex cells" has shown individual responses for specific visual stimuli of certain appearances within a certain *receptive field*. As an example, simple cells in the first layers fired for low-level features like edges or gradients, whereas complex cells in deeper layers got excited by more abstract higher-level features, such as shapes or oriented motion. It seems, the imitation of such a hierarchy fairly contributes to the success of deep neural architectures in computer vision.

However, such deep learning approaches do not have benefits only. The training procedure of hierarchical and convolutional architectures is not trivial. First of all, a proper network design is of great importance for success. This comes with tuning numerous critical parameters, such as the number of layers, the number of neurons, weight initialization, learning rates *etc.*, which is done manually. This is challenging for most applications and a lot of experience is required. Further, a massive amount of data has to be provided to train deep systems. Even more, all the data needs to be labeled to train supervised models [70]. As a consequence, preparing training data of sufficient size is usually expensive and time-consuming. Convolutional architectures also require a vast number of weights or parameters to be trained, at high computational cost, however, unsupervised pre-training of network weights has shown to circumvent initialization and speed up training progress [29]. In this context, parallel computing is essential for acceptable deep learning time periods, though training still requires hours or even days for big and complex structures.

Also during test time, deep convolutional architectures need to perform a large amount of convolutions. In practice, however, computational efficiency is especially important, in particular for mobile applications. Many real-world applications require real-time processing where costly operations are unfavorable. To circumvent the mentioned disadvantages, developing more efficient systems is of special interest.

The works of Coates *et al.* [20, 22] show that also efficient single-layer architectures enable relatively high image classification performances, although of being learned without supervision. They apply patch-based dictionary learning by using a standardized K-means clustering approach. Resulting centroids are then used as a dictionary of filter kernels that describe certain image characteristics on patch-level. In order to produce image representations, Coates *et al.* use this dictionary as a *feature extractor* by computing the inner product between learned filter kernels and image patches. Hence, image patches are represented as *code-vectors*, thus yielding feature channels for the entire image. Note that this corresponds to convolutional feature extraction, where a patch can be regarded as receptive field. After applying a simple non-linear activation on each code-vector, and performing spatial feature pooling, they obtain surprisingly well discriminative image representations, being competitive with a 2-layer Convolutional Neural Network [53] on classifying CIFAR-10 images

[52]). Remarkably they achieve this with linear classification (SVM). However, state-of-the-art approaches [19, 40] that use deep networks fairly outperform the single-layer architecture of Coates *et al.* Still, single-layer networks are of certain interest since they can be learned in a more efficient way and do not need dozens of layers for computation. Since Coates *et al.* use unsupervised learning, their approach bears an important additional advantage for the case when labeled data is scarce.

To recap, the approach of Coates *et al.* shows two major advantages: The *unsupervised* dictionary learning by simple K-means clustering on image-patches, and the *single-layer* feature extraction. However, to achieve high classification accuracies with such image representations, preprocessing parameters turned out to be critical. Regarding dictionary learning and feature extraction, especially standardization and whitening on patch-level have a great impact on the final classification performance. Further, this method still requires relatively time-expensive convolutional feature extraction.

To overcome above mentioned problems, we propose learning representations with Random Forests (RFs) [15, 16, 45]. Inspired by the approach of Coates *et al.* [20], our goal is to omit convolutional encoding by using an ensemble of independent binary decision trees to infer patch representations. A RF provides a more efficient way to obtain feature information, since we only have to traverse several decision trees whereby each tree performs fast binary pixel tests. In addition, our method allows a *direct* mapping of *raw* input patches to feature-space in a pixel-wise and patch-based fashion, which efficiently yields image representations. Due to this, also data preprocessing becomes obsolete at test time.

We evaluate patch-based representation learning with RF by investigating two different training approaches in this thesis. In our first approach we exploit the dictionary learning methods of Coates *et al.* [20, 21] where we either apply K-means clustering (KM) or Sparse Coding (SC) [21, 67] on a pool of random image patches. As a result, patches are assigned code-vectors that underlie the found dictionary. This code-vectors or patch-encodings are regarded as *pseudo*-label information. Hence, we use this pool of raw and pseudo-labeled patches to train our RF, enabling us to infer patch representations at test time.

Motivated by this approach, we examine a second, rather unconventional RF training methods. Recent works on CNNs show state-of-the-art results in various computer vision tasks [19, 53, 57], however, they suffer from high computational complexity. Many CNN architectures carry out relatively time consuming convolutions in the first-layer where filter kernels are usually large. Also a high number of filter kernels affects processing time. This encourages us to answer the following question: Is it possible to improve a CNN's performance by replacing the first linear layer by a non-linear RF? Therefore, we introduce a "hybrid" network which employs a RF as a first-layer feature extractor

instead of performing a linear convolution. Therefore we propose to employ a RF in the same way as in our first approach. However, since the hybrid network is optimized iteratively in an end-to-end fashion, we additionally have to consider how to "update" a non-differentiable RF. A solution to that is to iteratively add trees to the RF model, which subsequently compensates the previous ensemble error (also known as Gradient Boosting [34, 35]). This way, the RF and the remaining CNN layers are trained simultaneously, that is, we are learning first-level representations with supervision.

1.1 Image Representations for Image Classification

As global task we perform image classification since it is one of the most prominent tasks to evaluate image representations. Figure 1.1 illustrates a typical image classification pipeline: After data preprocessing, image features are extracted by encoding relevant information. Additional post-processing like spatial pooling yields more compact and robust image representations [13] that ideally allow easy distinction of image categories. Finally, a trained classifier infers an image class c from an unseen image representation $\mathbf{f}$, by using the most likely category as final prediction (in Figure 1.1 this equals to the class "bowling - indoor" $c = 3$).

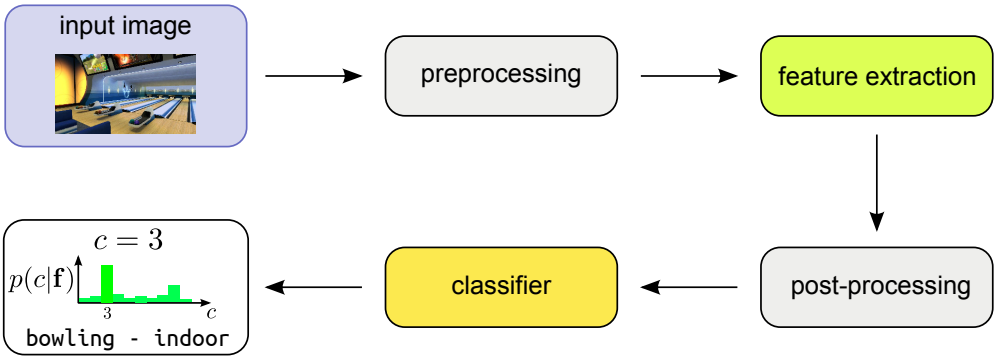


Figure 1.1: Overview of a typical image classification pipeline. An unseen image is classified using the most plausible category.

In this thesis, we investigate two different representation learning approaches that enable building image representations with RFs. Thus we omit otherwise necessary data preprocessing during feature extraction.

1.2 Unsupervised Representation Learning

Our first approach is inspired by the work of Coates *et al.* [20] who apply patch-based dictionary learning on random image patches. Hence we consider using two different dictionary learning algorithms, K-means clustering (KM) and Sparse Coding (SC) [21, 67]. As a result we do not only obtain a dictionary of filter kernels but also generate patch

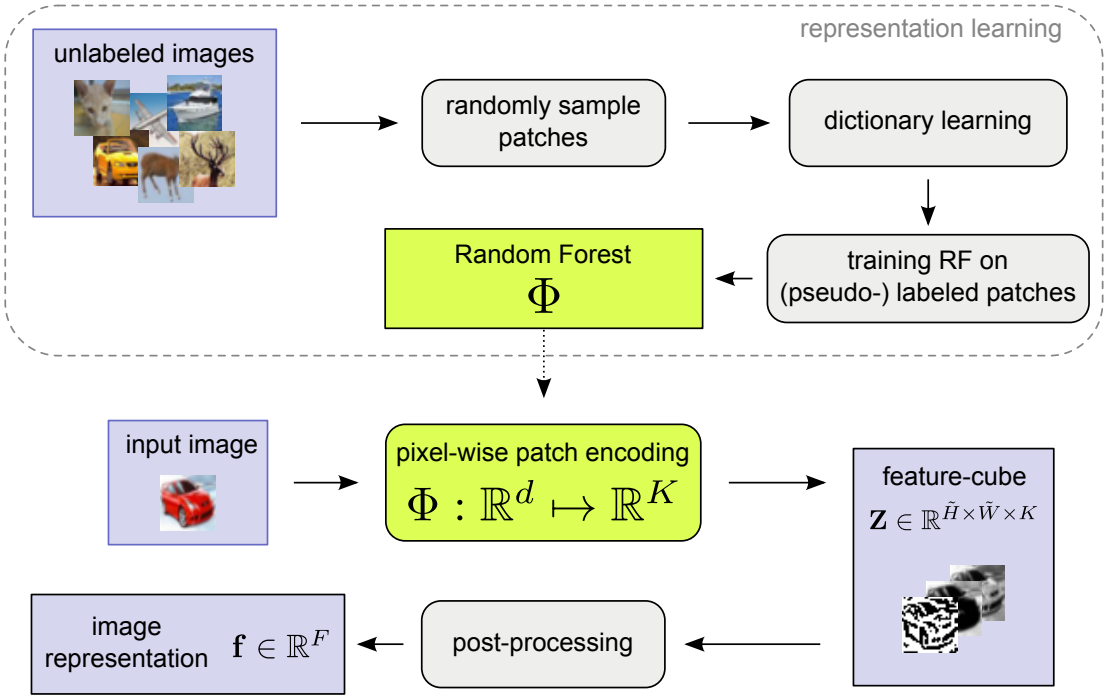


Figure 1.2: Overview of unsupervised representation learning with a Random Forest. The (pseudo-) label information of random patches is based on unsupervised dictionary learning, which we use to train a Random Forest. The trained Random Forest ($\mathbf{\Phi}$) is then applied to encode raw image-patches to feature-space $\mathbb{R}^K$ in a pixel-wise manner, yielding K feature channels. After post-processing we obtain a final feature vector $\mathbf{f} \in \mathbb{R}^F$ which represents the whole image.

pseudo-labels. More specifically, we learn *code-vectors* for image patches in an unsupervised manner where the underlying dictionary could be used to sparsely reconstruct the original image patches.

We use these random and pseudo-labeled patches as RF training data, which enables us to learn a direct mapping relation $\mathbf{\Phi}$ from raw input patches to a new patch representation in feature-space. Figure 1.2 illustrates how we propose to train and apply a RF to produce image representations.

To produce an image representation, we perform pixel-wise patch encoding across the whole image. Unlike convolutional feature extraction with filter kernels, this method is more efficient since we omit performing convolutions and take advantage of fast binary decisions of a decision tree ensemble.

In more detail, each patch of size $d = w \times w \times q$ (where a patch may have input channels $q \geq 1$) is encoded to a K-dimensional vector by RF inference. Hence, all resulting code-vectors of an image produce K feature channels, forming a "feature-cube" $\mathbf{Z}$. After we apply some post-processing (e.g spatial pooling), we obtain a final image representation by stacking all remaining features to a vector $\mathbf{f} \in \mathbb{R}^F$. Therefore, resulting image representations yield a classification error of 0.54% on the MNIST-10 dataset [56], using

60000 train-images and 10000 test-images for non-linear classification, where we use 10^6 raw image patches of size 6×6 with $K = 264$, on which we train 57 trees. We achieve this high performance within 12 minutes on multi-core systems, including unsupervised representation learning, as well as classifier training and testing. Further we show that our RF-based representations can improve experiments of the VLFeat Team [77] by simply appending them to existing representations.

1.3 Hybrid Feature Learning

In contrast to the previous representation learning method, this approach is based on a rather unconventional multi-layer architecture. We therefore propose to replace the first convolutional layer of a CNN by a Random Forest which we denote as *hybrid network*. Hence, the RF (again) encodes raw image patches in a pixel-wise fashion, and thus efficiently extracts first-level features. However, this requires additional considerations regarding network end-to-end optimization, since we have to train a RF and CNN layers simultaneously.

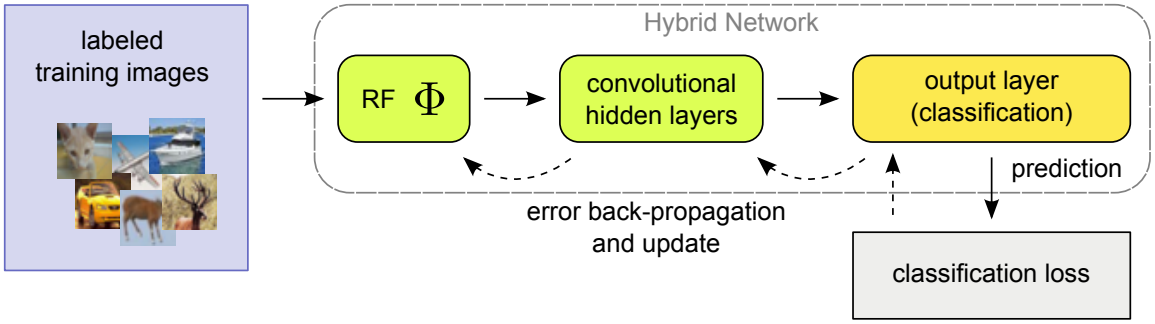


Figure 1.3: Overview of a hybrid network end-to-end optimization. A Random Forest is used to extract first-level features whereas a subsequent CNN derives further feature-levels up to a final class prediction. The resulting network error can be back-propagated up to the RF output. Training a new decision tree on this RF output gradient allows to "update" the RF encoding.

In order to train a hybrid network, we iteratively back-propagate the output-error through all layers, update the network weights and train the RF in a boosting-like procedure. The reason for this is that end-to-end optimization requires network layers to be differentiable. If we replace the first layer by a RF, this RF-layer is not differentiable with respect to its model parameters (like split-functions, thresholds, etc.). Thus, an "update" of a RF in this sense is not viable. Instead, an ensemble of decision trees allows additive modeling [35] throughout optimization. Therefore we back-propagate the network error up to the RF output, where the RF output gradient is equal to so-called ensemble residuals. During optimization, the goal is to compensate the RF error by training a new tree on the previous ensemble residuals (Gradient Boosting [34]). Note that this corresponds to supervised representation learning as we have to compute the network error (or loss)

which relies on the given ground-truth information. Figure 1.3 illustrates how to optimize such a hybrid network in an end-to-end fashion.

To prove our concept, we again examine network performances on image classification and compare a baseline CNN with a hybrid version of it. Experiments on MNIST reveal, that a first-layer RF with a training depth of 8 improves the network's classification performance by approximately 17%, where we achieve a classification error of 1.49%. We explain this improvement by the non-linear nature of our RF which enables learning combinations of low-level features. Further we address different aspects on RF initialization to ensure proper network excitation.

1.4 Synopsis

In Section 2 we discuss global preliminaries and give a brief overview of used learning methods as well as related works on feature learning. Further we explain how to build a RF and thus how to train and test binary decision trees in general. The RF feature extraction is stated in Section 3 which is equal for both our approaches. In Section 4 we explain our first approach that employs unsupervised representation learning. We examine the impact of various learning parameters as well as RF specific parameters on overall classification performance at test time. Therefore we show various experiments on popular benchmark datasets like MNIST or CIFAR. Our second approach is stated in Section 5 and introduces a baseline CNN as well as a hybrid network. We evaluate both networks on MNIST classification and examine how RF parameters affect the overall network performance. Finally, the conclusion of our work is summarized in Section 6 where we also discuss ideas for future research.

2 Preliminaries and Related Work

In this section we address preliminaries, as well as related work including methods for image feature extraction or encoding. Further we discuss existing representation learning approaches where we disregard whether the actual application of the discussed works was image classification.

In computer vision the goal is to analyze visual data, enabling systems to "understand" images. Therefore, extracting descriptive image characteristics, also called image features, is crucial [4, 11, 24, 73, 80]. Already plain pixel intensities hold certain information although they belong to the lowest level of features. For example [59] addresses a simple approach that just uses color-histograms for sufficient object tracking. Besides that, more sophisticated methods use manually designed features which have been (and still are) widely used for various computer vision tasks [11, 77].

Such popular hand-crafted representations encode low-level image information: As an example, the HOG-descriptor [25] encodes oriented image-gradient information over certain image regions to form orientation-histograms. Another commonly used feature descriptor is known as SIFT [58], where local regions of an image are considered, again involving gradient information. The strength of SIFT is its invariance to scale and rotation. A global image descriptor which tries to capture entire scene characteristics is denoted as GIST-descriptor, introduced in [5]. It uses spectral information by applying Gabor-filtering which characterizes dominant spatial structures of an image. In contrast, Ojala *et.al.* [66] introduce Local Binary Patterns (LBP) which are capable for encoding textural image information. Therefore, they regard a certain neighborhood-pattern around a center-pixel where value comparisons are captured in a binary form. LBP outline powerful features especially for classifying textures in image-data [66].

Note that these are just a few successful and widely used hand-crafted feature concepts. This also encourages us to compare the discriminative power of common hand-crafted features to our representations with respect to image classification accuracy.

The VLFeat-Team [77] examined image classification using multiple types of features or descriptors that are concatenated to form an image representation. Their experiments on different benchmark datasets show, that stacked representations allow higher classification rates than if using single representations. Hence, each kind of representation may encode different characteristics. Therefore, our idea is to enhance their classification pipeline by appending our RF representations to their feature stack. We will show that this slightly improves their results.

Even though extensive research had been dedicated to hand-crafted features, they often cannot compete with recent methods which employ *learned* features [4]. Another drawback of using multiple traditional feature encodings is that each of them may have numerous parameters that have to be tuned. Chatfield *et.al.* [18] show how difficult it is to compare feature encoding algorithms like the *Kernel codebook encoding* [36], *Fisher encoding* [68], *Super vector encoding* [83] and many more. Quite a few properties and parameters for each of these methods have to be chosen or tuned wisely to work well, as small variations influence the final representation. This points out the complexity of advanced feature encodings that try to gather data statistics. As a consequence, the interest on feature learning or representation learning has grown over the last decade [20, 22, 41, 51].

Before we further discuss representation learning approaches, a general overview of relevant machine learning algorithms is given in the next section.

2.1 Overview of Learning Methods

Computer Vision generally involves various machine learning algorithms. However, the two approaches in this work only differ by two main machine learning concepts, *supervised learning* and *unsupervised learning* [62]:

Supervised learning always requires label information which can also be seen as having a “teacher” telling the ground-truth. In contrast, data without any label information is fed to unsupervised learning methods. In this case, one aims on searching patterns, structures or groups of data points to encode or represent them in a new “label”-space.

Considering our first approach, we make use of unsupervised dictionary learning on random patches. Hence, we do not use any label-information. This also means that we learn our patch encoding scheme without supervision, although training a Random Forest technically belongs to supervised learning.

Our second approach strictly uses ground-truth labels of images to determine a loss, which is used to optimize a multi-layer architecture. Hence, supervised end-to-end optimization allows to learn features within network layers, which also applies to training a first-layer Random Forest.

2.2 Related Work on Representation Learning

In the last few years, many works on representation learning appeared, utilizing different machine learning concepts [21, 43, 65, 67].Particularly worth mentioning is the review on representation learning of Bengio *et al.* [7] who compare a variety of approaches in this field of research.

Nowadays, state-of-the-art image recognition is generally achieved by deep learning architectures [19, 57, 75, 76] that use multiple layers, thus enabling to learn representations on different levels of abstraction. The vast majority of these successful networks are based on convolutional layers, where in-between spatial pooling layers are used to reduce dimensionality and induce scale invariance. Although each layer may have a linear activation, the deep hierarchical structure is the key to learn high-level features.

Examples of success using deep learning networks can be seen in the work of Ciresan *et al.* [19]. They introduce a committee of several Deep Neural Networks (DNNs) forming a highly complex ensemble model. In addition, each DNN is trained independently using differently transformed input images which tremendously enhances the variety of training data (data augmentation), facilitating high generalization. Hence, their DNN committee shows state-of-the-art results on various popular benchmark datasets, e.g. they achieve a classification-error of only 0.23% on the MNIST dataset [56]. This low error-rate can be explained by the ensemble strength of using several independently trained DNNs.

However, a drawback of such multi-layered architectures is the massive amount of required network weights that have to be initialized and optimized. To speed up network training, unsupervised learning methods exist that allow to pre-train network layers [29], e.g. by using dictionary learning approaches [20, 67] or auto-encoders [44].

Vincent *et al.* [79] show that auto-encoders are capable of extracting robust features, that even can be used for multi-layer initialization if stacked. Memisevic *et.al.* [61] also applied auto-encoders which facilitate learning image transformations. They train on pairs of transformed images and force hidden weights to encode affine transformations or artificial motion, illustrated in Figure 2.1. According to this, also within-image structures and correlations are learned by only using image-patches.

Concerning patch-based representation learning, Agarwal and Triggs [1] discuss the following drawback: If one learns a dictionary on image patches, only local appearances are captured. Thus, Agarwal and Triggs introduce a method that is capable to encode co-occurrence statistics at larger scale. By describing features in a multilevel structure using different patch-sizes, they iteratively gather local features and represent them in a more global scale. Finally they observe an increasingly abstract description of image feature content which shows to improve performance for many classification tasks.

Another unsupervised approach is to learn mid-level discriminative patches, proposed by Singh *et al.* [73]. They learn a set of discriminative patches at arbitrary scale, which

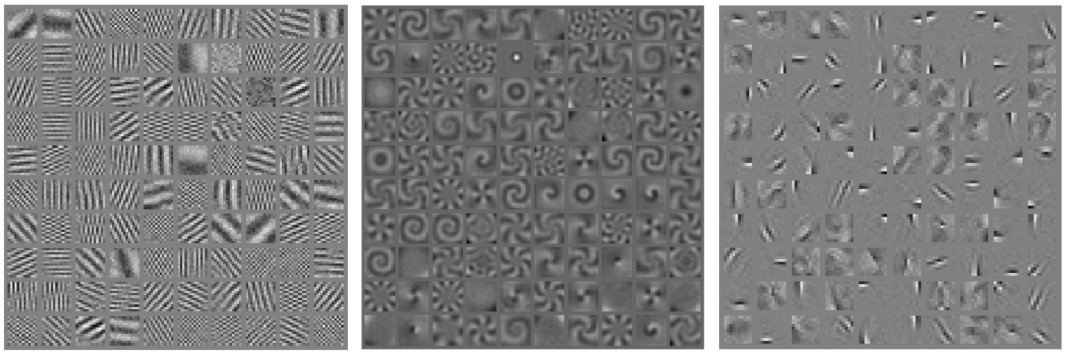


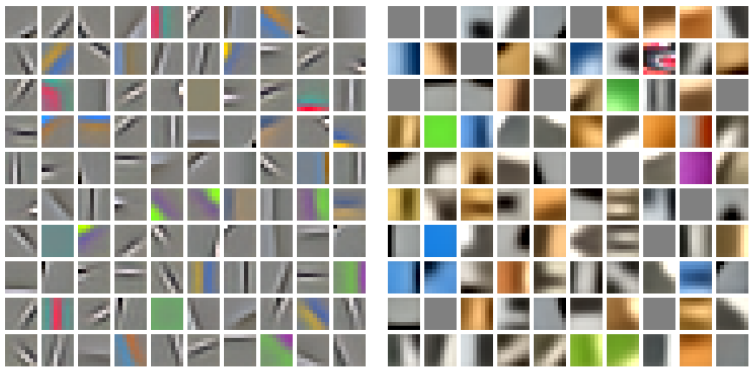
Figure 2.1: Examples of filters learned on image transformations by auto-encoders [61]: Translations (left), affine (middle) and motion discontinuities (right).

capture the characteristic image content with respect to the object class. For this, they only use the HOG descriptor [25] as input feature data.

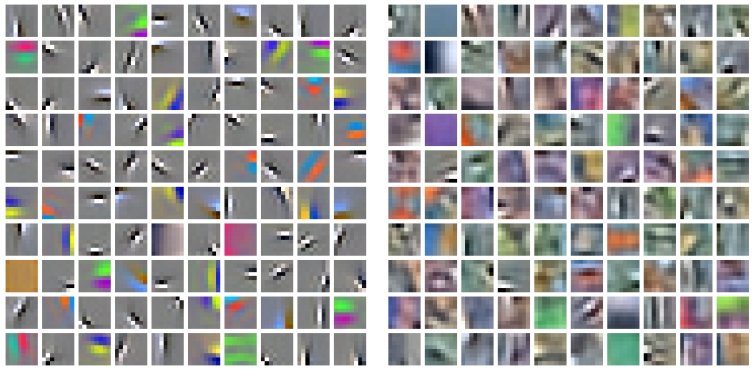
Zhang *et al.* [82] propose to enhance patch-based representation learning by only focusing on salient patches that contain most relevant information. Instead of using random or dense image patches, they employ a saliency guided sampling strategy. Hence, to find salient patches they use color information and position of neighboring patches by following a dissimilarity rule. Based on the resulting set of salient patches they learn a sparse auto-encoder [44, 79] in an unsupervised fashion. Resulting representations show to improve image classification results over [58] features.

Whether for pre-training layers of deep belief nets [29, 43] or for representation learning in general, patch-based dictionary learning approaches like K-means clustering or Sparse Coding (SC) [21, 67] have gained significant attention. Their simplicity as well as the fast feature extraction capability are main advantages over complex architectures. Although Sparse Coding prove to yield better dictionaries than K-means [21], the required training time for high-dimensional problems may become impracticable. Instead, dictionary learning with K-means clustering is faster and thus competitive to SC. Coates *et al.* analyzed such single-layer architectures and show to achieve surprisingly good image classification results [20, 22]. The above mentioned patch-based dictionary learning algorithms use random image patches [65]. To extract features, patches are encoded by a simple inner product with the learned dictionary filter kernels which finally yields image feature channels. A following non-linear thresholding on patch encodings discards weak features and yields more sparse representations. To obtain final image representations, Coates *et al.* additionally perform spatial pooling (see Section 2.3) within the feature channels. Although they evaluate their representations on linear classification, they show to even outperform a 2-layered Convolutional Deep Belief Network [53].

However, Coates and Ng state in [22] that it is not trivial to get such a single-layer architecture to work well. Especially preprocessing steps like normalization and whitening



(a) K-means (with and without whitening)



(b) Sparse Autoencoder (with and without whitening)

Figure 2.2: Visualization of learned dictionaries on CIFAR patches [20].

[48] of patch data appears to be critical, since they directly use the found dictionary of filters during feature extraction. Examples of found dictionaries can be seen in Figure 2.2 (Coates *et al.* [20]), where they visualize the influence of whitening. Also proper post-processing of patch encodings as well as spatial pooling is of great importance. Hence, their good performance may primarily come from wisely chosen parameters across the whole classification pipeline and high dimensional (overcomplete) representations.

To recap, the single-layer architecture proposed by Coates *et al.* is highly sensitive to data preprocessing and extracts features by convolving the input image with learned dictionary filters. In this sense, large filters or large dictionaries have negative properties regarding processing time during feature extraction. In contrast, our approach uses a RF to directly infer patch representations from raw input data. Therefore we provide a more efficient way to produce image representations (see Section 3).

Mossmann, Triggs and Jurie [63] also use Random Forests for feature extraction. They introduce Extremely Randomized Clustering Forests (ERCF) for bag-of-features image classification. Their idea is to directly use a ERCF as a visual dictionary whereby its output is a final bag-of-features description of an image. Therefore, randomly sampled

Figure 2.3: An illustration of obtained probability maps for the label 'bike' using ERCF [63]. Four different images (top) with corresponding probabilities shown for bikes as brightness (bottom).

image patches at arbitrary position and scale are encoded to a local descriptor-vector form. The underlying image label is used as patch label information. Based on this, a ERCF is trained, separating the descriptor-data. At testing time, they feed descriptors of densely sampled image patches to each tree and only use the resulting ensemble leafnode-index, neglecting the actual leafnode histogram. Therefore they accumulate over all forest leafnode indices to produce an index-histogram. By using this representations they obtain probability maps for certain categories, e.g. as shown for the label 'bike' in Figure 2.3.

Notice several differences to our approach: Instead of using raw pixels as input data they use feature-descriptor data which requires computation beforehand. Further, the ERCF does not utilize the leafnode histograms but accumulates the corresponding leafnode indices to form an index-histogram.

The related work of Shotton *et al.* [72] introduces Semantic Texton Forests (STFs) for image classification and segmentation. Similar to our approach, they train an ensemble of random decision trees on raw image patches. During training, powerful texton codebooks are learned and semantic textons are implicitly clustered by a tree. This is exploited during feature extraction as they concatenate all histograms of all traversed ensemble nodes. They found that this representations give poor but still surprisingly good image classification rates regarding pixel-level classification only [72]. However, they also form more powerful features by additionally involving image region statistics of resulting semantic textons. Hence, they also compute a region-prior by averaging across all STF leafnode distributions within an image region. By learning with supervision, this allows to infer certain low-level semantics. Shotton *et al.* further show that the histograms and region-priors are complementary, and that hierarchical concatenation works better than if concatenating only terminal leafnode histograms. In contrast to their work, our first methods follows an unsupervised representation learning approach as we train a RF on random pseudo-labeled patches. Therefore our RF enables to infer patch encodings from raw pixel information by computing the average leafnode histogram of the ensemble.

2.3 Feature Pooling

At next we discuss spatial feature pooling, which usually improves the discriminative power of final image representations [13].

Today, spatial *feature pooling* is a common method among modern image recognition approaches and usually have a big impact on final performance. Effects are increased invariance to image transformations (like translation) and decreased representation dimensionality. Also the influence of noise and clutter in feature-space is damped. Pooling can also be described as gathering spatial statistics over image regions or patches. This reminds of the discoveries of Hubel and Wiesel [46, 47] where complex cells of the human vision implicitly perform pooling among overlapping receptive fields.

Before discussing different pooling operations let us first look at the principle of spatial pooling which is shown in Figure 2.4. Assuming that we have K feature-channels, we perform pooling within each channel over a spatial grid e.g. of size 2-by-2. Then, a certain pooling operation yields 2×2 values for each channel. Depending on the cell-size and stride, this may massively reduce the overall dimensionality of the final image representation.

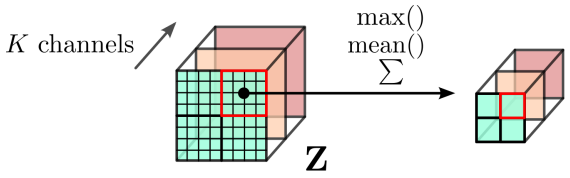


Figure 2.4: Spatial pooling of feature channels using a 2-by-2 grid. An activation function (e.g. max-pooling, avg-pooling or summation) is applied on each cell which reduces dimensionality and induces invariance.

Different pooling operations exist where the most common are max-pooling, avg-pooling or summation. *Max-pooling* is widely used and selects the maximum cell-value. This is equal to passing the most activated features and dropping all less important ones, including noise. Due to spatial cells, a maximum value may also vary its relative position in a certain range without affecting the final result. Thus, final representations show less sensitivity to image transformations such as translation or rotation. Further, max-pooling proved to work well for sparse features that have low probability of being active [13].

Another widely used pooling operation is a *summation* over pooling cells. In this case all features are involved, however, this may also include undesired noisy features or outliers which may counteract the discriminative power of representations.

Avg-pooling computes the mean value within pooling cells. This sometimes improves performance over max-pooling but it is rarely the case [13].

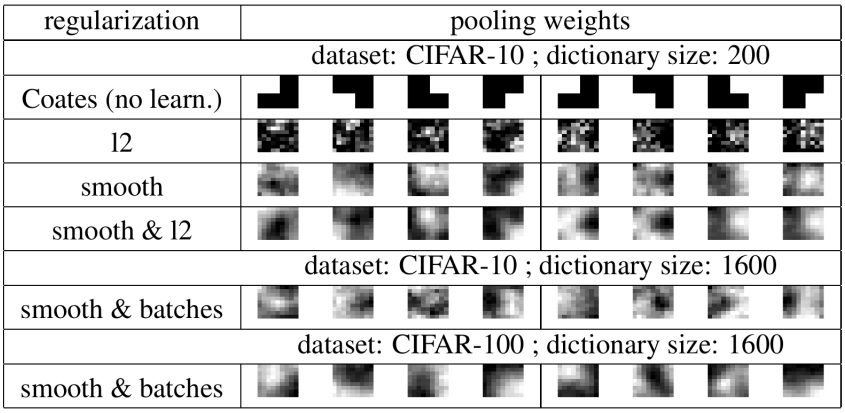


Figure 2.5: Visualization of pooling-weights applying different learning strategies and regularizations. Results are shown for varying datasets and dictionary-sizes. Brighter regions denote larger weights. Top row shows fixed pooling cells. [60]

A theoretical analysis of common pooling operations has been done by Boureau *et al.* [13]. One conclusion of this work is that the choice of a proper pooling operation highly depends on the actual application as well as on the preferred step-size (stride). Many works exist addressing these problems [11–13, 36].

Due to the fact that spatial pooling has big impact on the output, also undesired effects may occur. An unfavorable characteristic of pooled features is discovered in the work of Jia *et al.* [51]. They discuss possible redundancy within representations after spatial pooling is applied on feature channels. Therefore they refer to the method of Coates *et al.* [20] who apply patch-based K-means clustering to learn a dictionary of filter kernels. This often yields several Gabor-like filters at same orientation, which only differ in small translations. Such small differences cause quite similar patch encodings, and if we now apply spatial pooling, these differences are lost resulting in redundant data within final image representations. Hence, Jia *et al.* propose to find better filters by learning a larger dictionary beforehand. They compute covariances between filters and perform feature selection [33] to finally obtain a dictionary of original size but with less correlated filters. However, concerning our approach we do not use the learned dictionary but use the underlying patch code-vectors to train our RF. Thus, during feature extraction our RF encodes image patches where the actual output is not directly dependent on the learned filter kernels.

Another interesting approach was published by Malinwosky and Fritz [60]. Instead of using fixed pooling regions, they learn smooth pooling regions, where examples are illustrated in Figure 2.5 [60]. They introduce pooling weights over feature channels and learn them simultaneously with classifier parameters. Using standard gradient descent, they achieve improved results compared to Coates *et al.* [20], especially for small dictionary sizes. They show that variable and smooth pooling regions are able to adapt to salient

areas in feature-space, emphasizing relevant subregions. However, in this work we follow Coates *et al.* and use fixed pooling regions by using summation-pooling as well as max-pooling, where the best choice depends on input data and the actual architecture.

2.4 Image Classification

As overall goal in this work we perform image classification on several popular image datasets. That is, we aim for correctly classifying unseen images using a trained classifier model. This allows to evaluate the discriminative power of obtained representations by examining resulting classification accuracies. However, to train a classifier, we first need suitable training-data that ideally provides the most discriminative characteristics of images.

Assuming a feature extractor that maps an image $\mathbf{I} \in \mathbb{R}^{H \times W \times q}$ to a feature vector $\mathbf{f} \in \mathbb{R}^F$, the goal is to setup a global dataset in feature-space. Note that in this context a feature vector denotes an image representation.

A given dataset $\mathcal{I}$ of raw images is usually split into a training-set $\mathcal{I}_{train}$ and a test-set $\mathcal{I}_{test}$, where each subset contains a certain number of samples N_{train} and N_{test}, respectively. Encoding all raw images of $\mathcal{I} = \{I_n\}_1^N$ gives a new dataset $\mathcal{F} = \{f_n\}_1^N$, where a "data sample" is defined as image or representation with its label c_n:

$$I_n = (\mathbf{I}_n, c_n) \tag{2.1}$$

$$f_n = (\mathbf{f}_n, c_n) \tag{2.2}$$

In this sense only image data is mapped to feature-space while the ground-truth label information $c_n \in \{1, \ldots, C\}$ is inherited. This further allows to train a classifier-model with supervision on the training-set $\mathcal{F}_{train}$. We introduce a classifier ζ that enables distinction between unseen samples. Thus, we aim at correctly predicting the ground-truth class c_n of test images, based on the extracted feature vector ($\zeta(\mathbf{f}_n) = \hat{c}_n$). Therefore the resulting image classification performance depends on the classifier model *and* the extracted image representation quality. However, in this work we put the main focus on obtaining powerful image representations $\mathbf{f}_n$ instead of optimizing classifier training. Hence, we fix classifier training parameters and only examine the final classification results.

If a classifier ζ has to predict more than two categories ($C > 2$) it is denoted as multiclass problem. There exist several algorithms who naturally handle multiclass tasks, like neural networks [3], k-Nearest Neighbor algorithms [2] or decision trees [16].

However, considering our first approach we decompose this multiclass problem into binary classification problems [10]. Hence, a decomposed multiclass problem uses an ensemble of binary classifiers which may be trained in a one-versus-all fashion. The drawback

of this kind of learning is that each one-vs-all-classifier sees a different and unbalanced sample distribution which may affect final prediction accuracy [8]. Nonetheless, we follow this popular approach since it is more efficient and we more or less disregard the classifier model. Thus we define ζ as multiclass classifier that employs one-versus-all binary classifiers H_c. Each of these binary classifiers returns a specific class-score where the final prediction is regarded as the most plausible category $\hat{c}$:

$$\zeta(\mathbf{f}_n) = \underset{c}{\operatorname{argmax}}\, H_c(\mathbf{f}_n) = \hat{c_n} \Big|_{c\in 1...C,\ \mathbf{f}_n \in \mathcal{F}_{test}} \tag{2.3}$$

For image classification we consider using two different models. The well known discriminative and linear SVM [10] basically fits a hyper-plane into feature-space that separates the training samples into two disjoint groups. Therefore, the optimization algorithm maximizes the margin on specific samples, called Support Vectors. This can be done very efficiently since this can be formulated as a convex optimization problem.

As second classifier we employ a non-linear classification-forest. It is trained by Adaptive Boosting (adaBoost) [32, 35, 42], which is an ensemble-learning method that adaptively trains the set of decision trees. This approach applies stage-wise additive modeling [35] of 'weak' trees which results in better RF performance. In addition to that, each training-sample is iteratively weighted, according to their contribution to the overall performance. Thus, we are able to distinguish between hard and easy samples, which allows us to emphasize them individually during training.

2.5 Random Forests

This section addresses training and testing an ensemble of decision trees, also known as Random Forest (RF) [15, 16]. In general RFs proved to be applicable for various tasks and are easy to implement. They provide fast training and testing by simple data-thresholding. Due to their good generalization, RFs rank among the most successful machine learning algorithms, since typical applications are multi-class prediction [9], regression [16] or data clustering [63]. A big advantage is the capability for massive parallelization in training and testing, because individual trees are independent [27]. However, due to the recent success of multi-layered deep learning architectures, RFs are loosing some popularity. Nonetheless, they are still powerful predictors when computational resources are critical or efficient processing is essential.

The RF model belongs to Ensemble Learning [27] and exploits the strength of multiple "weak" estimators. That is, a trained RF consists out of several randomly trained and independent identically distributed binary decision trees. Due to randomized training, each individual tree may differ from others and therefore behave differently. As all tree responses are gathered during test-time, this enables producing a more robust global prediction that outperforms the individual tree estimations.

To ensure optimal ensemble performance, certain conditions have to be met: (i) each weak estimator has to have at least better accuracy than random guessing on unseen samples *and* (ii) an ensemble should hold sufficient diversity. With respect to RFs, condition (ii) is true for a certain degree of randomness within decision trees. Related to this, several training approaches exist like Bagging [14] or Boosting [35]. Bagging requires to train an individual estimator on a random subset[1] (Bootstrapping) which may be of smaller size. Hence, each estimator sees a different training data distribution which implicitly injects randomness and de-correlates weak estimators. Benefits are reduced over-fitting characteristics and a more robust ensemble prediction [14].

Boosting, assigns a confidence weight to each weak estimator according to its estimation-strength on the entire training-set. The final prediction takes these individual weightings into account and thus global accuracy is improved [35].

Basically above mentioned ensemble training approaches can be combined or applied in different ways. In this work we consider using Bootstrapping to get differing training-subsets per tree. Further we also subsample data in each split-node to ensure additional tree diversity [71]. The next sections address the training as well as the testing procedure of a RF in more detail.

2.5.1 Training a Random Forest

A RF can either be trained in a supervised [9, 14, 16] or in an unsupervised manner [63]. The latter case requires label information and stores certain label statistics in terminal leafnodes. Label-information may either be in vector or scalar form, however, regarding our approaches we assume label-vectors per training sample.

Given a *labeled* training-set $\mathcal{S}_{train} = \{(\mathbf{x}_m, \mathbf{l}_m)\}_1^M$ where M denotes the number of training-samples, each sample $\mathbf{x} \in \mathbb{R}^d$ is of length d and has a label-vector $\mathbf{l} \in \mathbb{R}^K$ of length K. During training a tree, we recursively split all training-samples assigned to a node into two subsets (binary decision) until a termination criterion is met. In that regard label vectors are used to evaluate the split.

At the beginning, the root-node splits the entire training-set $\mathcal{S}_{train}$ and produces two child-nodes with disjoint subsets. Henceforth, we denote $\mathcal{S}_i$ as the actual node-subset of node i which has to be split into two further subsets $\mathcal{S}_i^L$ and $\mathcal{S}_i^R$ assigned to either the left child-node or the right child-node, respectively. This is illustrated in Figure 2.6. Hence, we define $\mathcal{S}_i = \mathcal{S}_i^L \cup \mathcal{S}_i^R$ and $\emptyset = \mathcal{S}_i^L \cap \mathcal{S}_i^R$ as subset constraints. Following this, all nodes are recursively separating their samples and finally form a decision tree which is defined by split-nodes and leafnodes.

The recursive training continues until we either reach a maximum tree depth[2] or we come below a predefined number of minimum subset samples required for further splitting. If a node of a tree t is set as leafnode j, the label vectors of the leafnode-samples are

[1] In case of Bagging a subset is sampled with replacement.
[2] A maximum tree-depth limits the total number of possible leaf-nodes.

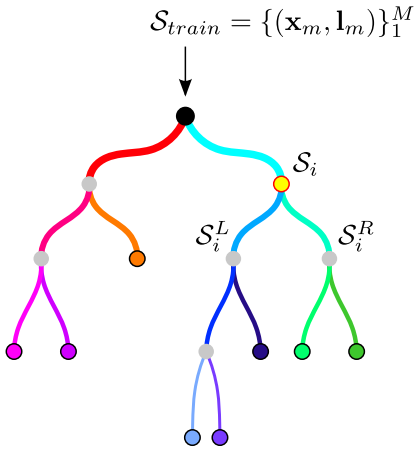


Figure 2.6: Training a decision tree; we split a node-subset and produce two child-nodes.

accumulated, producing a leafnode-distribution or histogram $p_j^{(t)}(k|\mathbf{x})$ over K bins, where $k \in K$.

$$\mathcal{S}_i^L = \{m \in \{1, \ldots, M_{S_i}\} \mid \Gamma_i(\mathbf{x}_m) < \tau_i\} \tag{2.4}$$

$$\mathcal{S}_i^R = \mathcal{S}_i \setminus \mathcal{S}_i^L \tag{2.5}$$

Split-Functions: During training, the goal is to split the dataset $\mathcal{S}_i$ of a current split-node i according to a random split-function Γ_i, and a random threshold τ_i, where we refer to Equations 2.4 and 2.5. Typical examples for these split-functions are e.g. using only a single variable $\Gamma_i(\mathbf{x}) = x_a$, a pair-wise comparison of two variables $\Gamma_i(\mathbf{x}) = x_a - x_b$, or advanced functions e.g. region based split-functions that use Wavelets [80], where $x_a, x_b \in \mathbf{x}$ and $a, b \in \{1, \ldots, d\}$.

To determine a good split we evaluate a few threshold-and-split-function pairs and choose the best performing setting as final node-split. Because we evaluate just a few tests, this ensures randomized training which reduces tree correlation within the ensemble. This further allows better generalization of the whole Random Forest model. As mentioned, we also consider applying Bagging [14] which contributes to this effect. Hence, for each tree we provide a Bootstrapped training-set $\mathcal{S}_{train}^{(t)}$ which is of the same size as the original training-set. In addition to that, we perform per-node sub-sampling where only a small subset of split-node training samples is considered for split-evaluation [71]. This massively speeds up tree training and again injects randomness.

In contrast, e.g. Extremely Randomized Trees [37] do not require any split evaluation during training since split-functions and thresholds are set totally random.

Split Evaluation: We randomly draw $\sqrt{d}$ split-functions Γ (e.g. for $\Gamma_i(\mathbf{x}) = x_a$ we choose $\sqrt{d}$ variables of an input vector), and test each one on n_{th} random thresholds. Thus, we have to evaluate $\sqrt{d} \times n_{th}$ splits by using a certain score. Therefore we compute an *information-gain* I which measures the quality of a split. Recall that an ideal split separates the node data to two disjoint subsets with respect to label information and thus maximizes the information gain. Assuming that we have a child subset $\mathcal{S}^h$ with label vectors $\mathbf{l} \in \mathbb{R}^K$, we regard each of the K elements as a discrete class or category. We then compute the information gain I for each split using the entropy within each of the two subsets, by following

$$I = H(\mathcal{S}) - \sum_{h \in \{L,R\}} \frac{|\mathcal{S}^h|}{|\mathcal{S}|} H(\mathcal{S}^h), \tag{2.6}$$

where the Shannon entropy is defined as $H(\mathcal{S}) = -\sum_{k=1}^{K} p(k) \log(p(k))$ with $p(k)$ being the occurring normalized class-frequency. In fact, the lower the entropy, the higher the information gain of the underlying split, causing final leafnode-histograms to be distributed more differently. However, in terms of efficiency we consider using the *Gini impurity* or Gini index $G(\mathcal{S})$ instead of $H(\mathcal{S})$, which is defined by,

$$G(\mathcal{S}) = \sum_{k=1}^{K} p(k)(1 - p(k)). \tag{2.7}$$

This way we approximate the Shannon entropy $H(\mathcal{S})$ which allows us to directly use the occurring class-frequencies $p(k)$ without the need for computing the logarithm. Hence, the information gain can be computed more efficiently. Note that regarding split performance, the difference to the entropy is negligible [16].

As we reach a leafnode during training, the final label-statistics of all assigned leafnode samples are gathered and stored in histogram form according to

$$\bar{\mathbf{l}} \in \mathbb{R}^K, \; \bar{l}_k = \frac{1}{M} \sum_{m=1}^{M} l_{mk} \quad \forall k \in \{1, \ldots, K\} \tag{2.8}$$

where we compute the average vector $\bar{\mathbf{l}}$ of all subset label-vectors $\mathbf{l}_m \in \{l_{m1}, \ldots l_{mK}\}$ of length K. This corresponds to the distribution $p_j^{(t)}(k|\mathbf{x}) = \bar{\mathbf{l}}_j^{(t)}$ which we also denote as leafnode histogram, where a leafnode $j^{(t)}$ belongs to a tree t.

In regard to the whole ensemble, a RF with T trees learns $\sum_{t=1}^{T} n_j^{(t)}$ leafnode histograms where $n_j^{(t)}$ is the total number of learned leafnodes within a single tree t.

2.5.2 Testing a Random Forest

In general, testing a sample $\mathbf{x}$ on a RF can be done very efficiently as we only have to traverse T trees. The meaning of the final output depends on the underlying training data

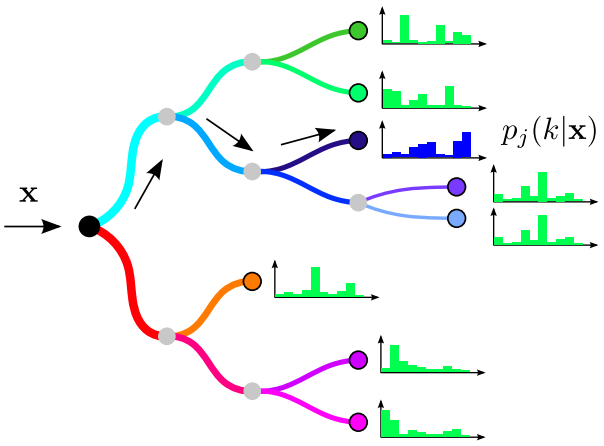


Figure 2.7: A sample is passed through a binary decision tree until a leafnode is reached. The tree returns an estimation $p_j(k|\mathbf{x})$ which we also denote as leafnode histogram.

which was used in training phase. During test-time each tree passes a sample according to its split-functions and thresholds to reach a certain leafnode j, which contains a histogram $p_j(k|\mathbf{x})$. As mentioned in the previous section, this histogram relies on the accumulated label-statistics of samples assigned to leafnode j during training. Figure 2.7 visualizes a sample-passthrough within a tree until a leafnode is reached. This way, each tree assigns a histogram to an input sample $\mathbf{x}$ where all T tree estimations are averaged to produce an ensemble prediction $\mathbf{z}$:

$$\mathbf{z} = \frac{1}{T}\sum_{t=1}^{T} p_j^{(t)}(k|\mathbf{x}) \tag{2.9}$$

If the RF is trained as a classifier, the output $\mathbf{z}$ equals a distribution over K categories, where the final prediction is the most likely category $\hat{c} = \text{argmax}_k\, z_k$.

However, this thesis primarily applies a RF as a feature extractor and therefore the purpose of an unaffected output $\mathbf{z}$ has important significance for ongoing data-processing. As stated in Section 3, we directly use the RF output as feature-data or more specifically as a patch representation.

3 Feature Extraction with Random Forests

The goal of this work is to investigate the feature extraction capability of a Random Forest model, which enables us to produce well discriminative image representations. This also means that the underlying RF training is of great importance. Considering that we address two different patch-based representation learning approaches in this work, the kind of label information that is used for RF training varies, which yields different RF output. For a comprehensible explanation of our RF-based feature extraction, we briefly explain the underlying meaning of the used label information.

Our first approach is based on unsupervised representation learning. In this case we apply patch-based dictionary learning that assigns sparse label vectors to image patches which we use as RF training data (for further explanation we refer to Sections 4.1.2 and 4.1.3). In contrast, our second approach uses label vectors that contain gradient information instead of being sparsely populated (for more detail we refer to Section 5.2.1).

Hence, both learning approaches provide image patches for RF training where the underlying label information enables to learn a mapping function, however, the meaning of RF inference is different.

Assuming that we already trained a patch-based RF, we aim at producing a discriminative image representation. We propose to do this in a pixel-wise manner, whereby such a RF allows to directly encode *raw* input image-patches. In more detail, we perform a non-linear and fast mapping of an image-patch $\mathbf{x} \in \mathbb{R}^d$ to a code-vector $\mathbf{z} \in \mathbb{R}^K$ in feature-space:

$$\Phi(\mathbf{x}) : \mathbb{R}^d \mapsto \mathbb{R}^K \tag{3.1}$$

In this sense, Φ is the mapping function of the RF. We therefore specify that an image-patch of size $w \times w \times q$ is available in vector form $\mathbf{x} \in \mathbb{R}^d$, where $d = q \cdot w^2$ and q denotes the number of input feature-channels, e.g. specific color-spaces or gradient maps. Note that commonly $K \gg d$ which results in a so-called overcomplete representation. Related to this,

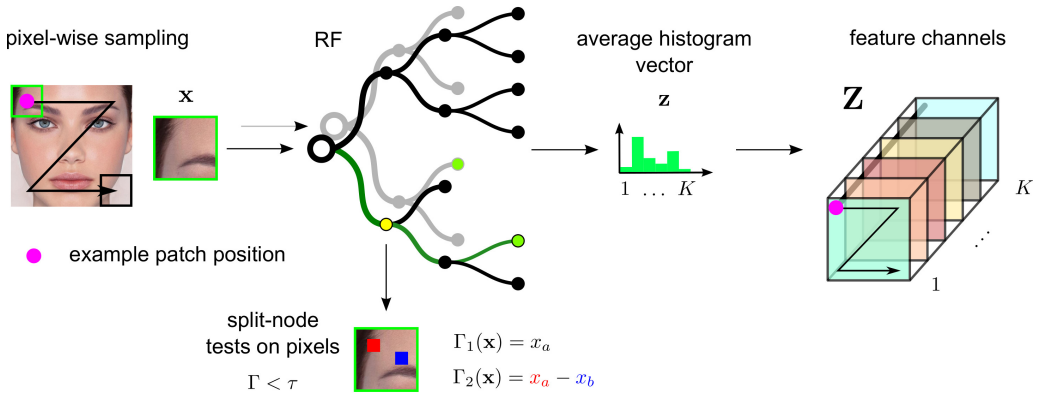


Figure 3.1: Visualization of pixel-wise feature extraction with a RF. Here, an already trained RF is applied. Each patch is traversed through the trees, passing several split-nodes which perform pixel tests according to a split-function Γ and a threshold τ. For each patch-location the RF returns an average histogram vector $\mathbf{z} \in \mathbb{R}^K$ where K corresponds to the patch representation length. As a result, we obtain a feature cube $\mathbf{Z}$ for the whole input image, where planes in depth show resulting feature channels. x_a or x_b are elements (pixels) in a patch vector $\mathbf{x}$ where $a, b \in \{1, \ldots, d\}$.

[22] show that overcomplete representations are especially suitable for linear classification.

Regarding feature extraction on entire images, Figure 3.1 illustrates how we propose to apply a patch-based RF. Therefore we aim at encoding image patches to produce a final image representation, which is similar to a bag-of-features model [24, 74]. Different patch sampling strategies exist, where recent studies show that dense sampling generally outperforms sparse sampling [65]. Hence, we consider applying the dense sampling strategy in terms of pixel-wise patch encoding (with stride = 1). In contrast to convolutional feature extraction, we just have to sample all required patches of an image and feed them to our RF. More specifically, each patch is passed through an ensemble of binary decision trees where each tree performs several pixel tests on input data. Therefore, a split-node i uses a certain split-function Γ_i and a corresponding threshold τ_i to assign a patch either to the left or to the right child node. If input patches are standardized beforehand, single pixel tests ($\Gamma_i^{(1)} = x_a$) may suffice to learn patch statistics. To enable appropriate decisions on unstandardized input patches, it is useful to perform at least pixel-pair tests ($\Gamma_i^{(2)} = x_a - x_b$) that allow learning within-patch pixel dependencies. Note, the indices a, b describe actual elements of the patch vector $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$, which correspond to certain within-patch pixels. Finally, the patch sample reaches a leafnode within each tree, where we average over all leafnode histograms, following Equation 2.9. Thus, the RF outputs an encoding of the patch sample. Applied on an entire image $\mathbf{I} \in \mathbb{R}^{H \times W}$ we obtain K dimensional patch encodings which yields a "feature-cube" $\mathbf{Z}$. In this context we regard $\mathbf{Z}$ having K feature channels or image representations. The mapping relation,

$$\text{pixel-wise RF-mapping:} \quad \mathbf{I} \in \mathbb{R}^{H \times W \times \mathtt{ch}} \;\mapsto\; \mathbf{Z} \in \mathbb{R}^{(H-w+1) \times (W-w+1) \times K}, \tag{3.2}$$

shows that the spatial height and width of such feature channels decrease compared to the input image, where $w \times w$ is the spatial image patch size. For illustration, Figure 3.2 shows some natural images of the STL-10 dataset [20] and examples of corresponding feature channels.

However, the following limitations have to be considered: In fact, a high number of feature channels K results in a very high dimensional image representation, which may become inconvenient for further processing. Hence, memory could be exceeded for a large image training-set during training a classifier model. To reduce dimensionality one may set a bigger stride during patch-based feature extraction on images. This speeds up extraction time as we have to encode fewer patches but usually decreases discriminative power of the final representation. In addition, spatial pooling over a certain grid across each feature channel (see Section 2.3) massively reduces data dimensionality and usually yields more powerful representations.

3.1 Time Complexity Comparison

In terms of time consumption, convolutional approaches and RF-based feature extraction behave differently. Let us assume q is the number of input feature channels and the input is of spatial size $H \times W$. Further, we assume that we have K convolutional filters with spatial size $w \times w$. Hence, we get a spatial output size of $(H-w+1) \times (W-w+1) = \tilde{H} \times \tilde{W}$. The resulting time-complexity of a convolutional feature extraction (one layer) is then given by

$$O(q \cdot w^2 \cdot K \cdot \tilde{H} \cdot \tilde{W}). \tag{3.3}$$

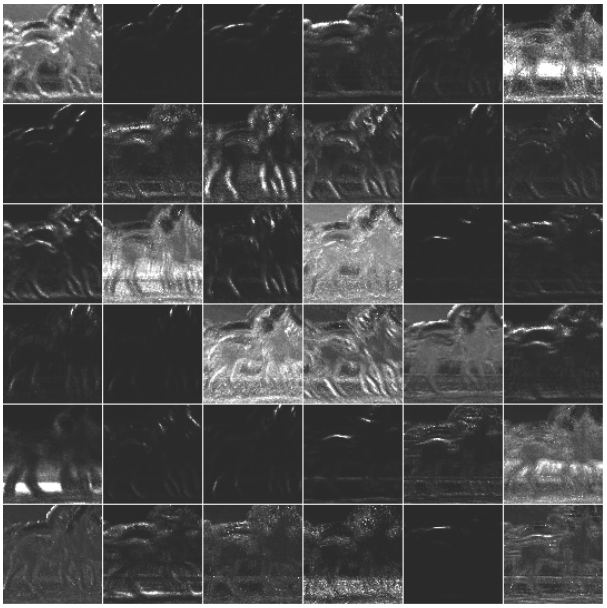
In contrast, if we perform pixel-wise patch-based feature extraction with a RF by using the same spatial dimensions and settings as mentioned above, we get a total time complexity of

$$O\Big(\tilde{H} \cdot \tilde{W} \cdot T \cdot D\Big), \tag{3.4}$$

where we assume T binary decision trees at a maximum depth of D. Recall that the total number of image patches depends on the patch-size w. Instead, the input dimensionality does not affect the theoretical time cost during testing because binary decisions are only made for input data, yielding fixed sized K-dimensional leafnode-histograms which can be obtained by lookup tables.

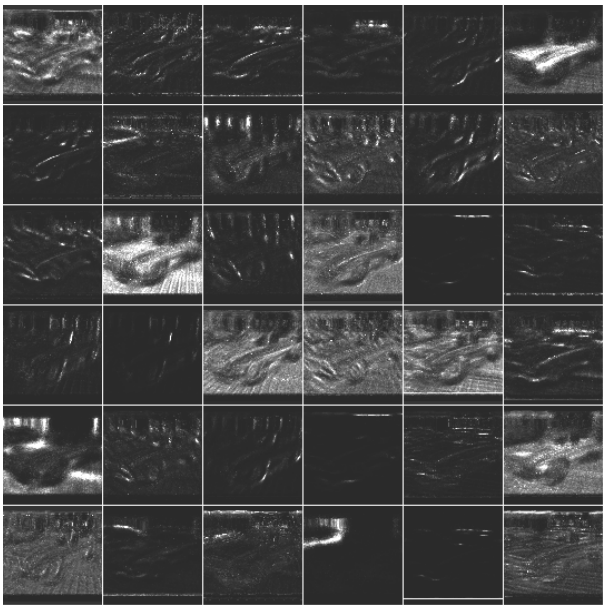
As seen, theoretical time complexity is reduced over convolutional approaches, especially for small RFs, big filters (w) or high numbers of input/output feature channels (q, K). Besides that, also optimizations exist that reduces time-complexity of convolutional processes in CNNs [23]. However, in this thesis we mainly focus on representation quality instead of examining efficiency.

(a)



(b)

(c)



(d)

Figure 3.2: Examples of feature channels extracted from natural images. Figures **(a)** and **(c)** show rgb input images where **(b)** and **(d)** visualize corresponding feature maps, respectively. Input images are part of the STL-10 dataset.

3.2 Implementation

We basically use the same implementation of our Random Forest for both proposed representation learning approaches. However, some RF training attributes vary.

On the one hand, we train a RF on pseudo-labeled patches using unsupervised dictionary learning (see Section 4). Here we examine two dictionary learning methods, K-means clustering (KM) and Sparse Coding (SC), which yield different patch-label information in terms of code-vectors. For KM we use one-hot encoding whereas SC outputs sparse code-vectors with positive or negative values. We assume these code-vectors as pseudo-label vectors where each vector element denotes a feature. Since we regard features as "classes", we train our RF as a classification forest (see Section 2.5.1). Therefore we apply the following split properties: in each split-node j we apply sub-sampling [71] and select 100 samples with replacement. Further we draw $d/2$ split-functions where d denotes the input vector dimensionality which equals the number of input pixels per patch. For each split-function we evaluate $n_\tau = 5$ thresholds. Finally, a split is defined by the split-function and threshold that shows best split properties (Equation 2.6).

On the other hand, our second approach (see Section 5) employs our RF implementation to do regression. In this context patch label-vectors describe gradients with negative and positive elements at any index. However, we still consider each vector element as a feature. The RF of this approach uses the following settings: we again apply sub-sampling within a split-node [71] and randomly choose 100 samples with replacement. But in contrast to the previous approach, we only perform $d/3$ split-functions with $n_\tau = 3$ tests, which in this case speeds up the iterative training process of a larger tree ensemble.

As seen, in both approaches patch label-vectors with positive *and* negative values can appear. Thus, the split-evaluation during RF training uses *absolute* vector values to compute an information gain using the Gini-index as stated in Equation 2.7. Hence, also negative vector entries contribute to split-decisions. Nonetheless, we produce leafnode histogram vectors regarding positive *and* negative values, allowing the RF to output negative and positive features. This is especially important if training on SC-based label-vectors or on gradients (error-residuals), which is similar to regression.

Note that all above mentioned settings are found empirically by making a tradeoff between efficient training and accuracy.

By reasons of efficiency, we implement the Random Forest in C++ using parallel computing utilities (OpenMP library). Because our frameworks are implemented in MATLAB 7.12.0 2011a, our RF implementation is build as a MEX-function which allows MATLAB to access C++ executables.

4 Unsupervised Representation Learning

In this section we investigate our first approach which addresses unsupervised representation learning with Random Forests (RFs). Inspired by the works of Coates *et al.* [20–22], we apply the following strategy to learn representations: First of all we exploit patch-based dictionary learning, which yields pseudo-label information in terms of patch code-vectors, to train a RF. This way, we learn an encoding scheme from raw image data to a new feature-space. To finally produce an image representation, we follow the procedure stated in Section 3 and encode all patches of an input image. Dependent on the actual application, additional post-processing on patch-encodings, like non-linear activation or spatial pooling, further improves the discriminative power of image representations. A pipeline overview concerning feature learning and extraction is illustrated in Figure 4.1.

To evaluate the RF-based representation quality, we consider performing image classification on common benchmark datasets and examine the impact of various parameters on overall classification performance.

The next section introduces preliminaries that explain unsupervised dictionary learning and how to train a RF as feature extractor.

4.1 Preliminaries

To train a RF for pixel-wise patch encoding, we at first need to generate proper RF training-data. Hence, we have to build a pool of image patches and assign label information. Since we apply unsupervised dictionary learning, we denote the resulting label information as *pseudo* label information. Note that the quality of this pseudo-labeled training set has great influence on RF inference. Figure 4.2 shows an overview of this process, using MNIST-10 images [56] as an example.

By randomly sampling raw image patches from a given input image-set $\mathcal{I} = \{\mathbf{I}_n\}_{n=1}^{N}$, we create a pool of M patches where we represent each patch in vector form $\mathbf{x} \in \mathbb{R}^d$. We introduce a patch-pool $\mathbf{X} \in \mathbb{R}^{M \times d}$ where a patch of size $w \times w \times q = d$ has q input feature

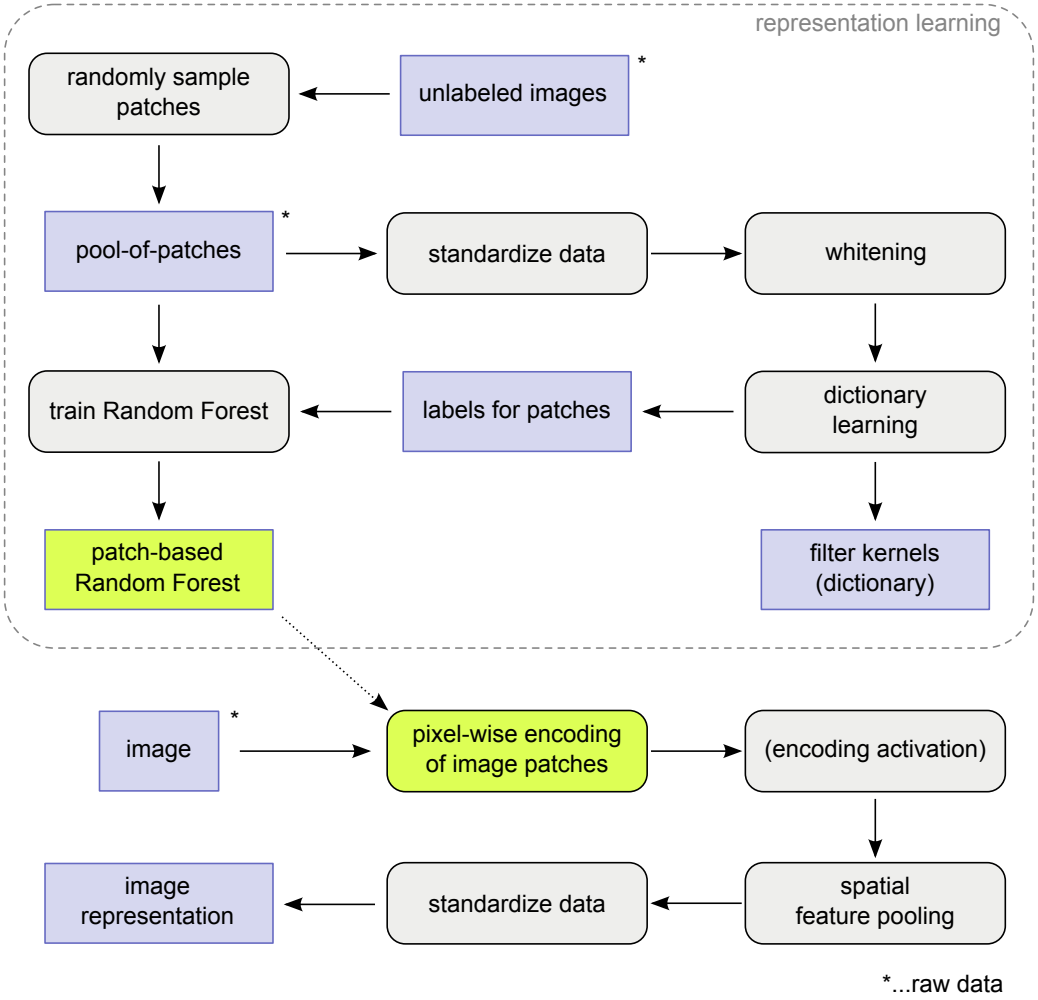


Figure 4.1: Proposed pipeline overview for unsupervised representation learning: (top) exploiting patch-based dictionary learning for RF training; (bottom) producing an image representation using patch-based RF inference and post-processing. *The raw data may have arbitrary input feature channels.

channels. Notice, that an image $\mathbf{I}_n \in \mathbb{R}^{H \times W \times q}$ may have an arbitrary number of input feature-channels ($q \geq 1$), such as rgb-images with $q = 3$ color-channels. Also gradient information like gradient-magnitude maps or oriented gradient maps are possible input feature channels.

The next step is to assign a pseudo-label vector $\mathbf{v}_m$ to each patch by using dictionary learning. Therefore we consider using two different dictionary learning methods, a modified version of K-means clustering (Section 4.1.2) and sparse coding (Section 4.1.3). However, to find good dictionaries, Coates *et al.* [20, 21] show that preprocessing on $\mathbf{X}$ is a critical step.

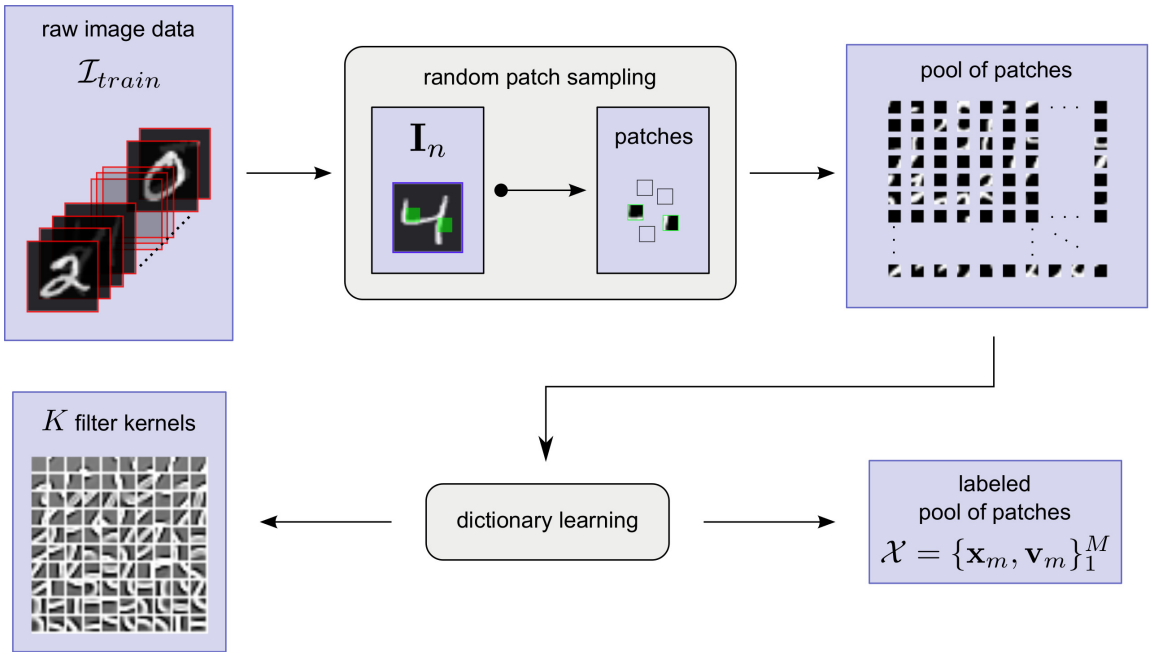


Figure 4.2: Dictionary learning on MNIST assigns pseudo label information to image patches.

4.1.1 Preprocessing for Dictionary Learning

Since we follow the dictionary learning methods of Coates *et al.* [20, 21], we also have to consider applying data preprocessing. Therefore best results are obtained if patches are standardized and whitened in advance.

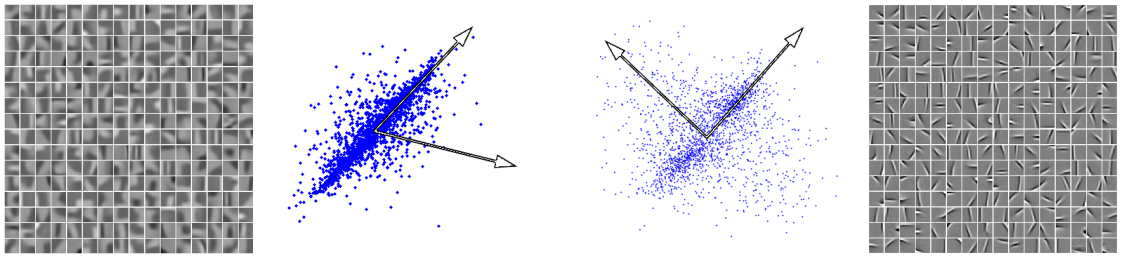
Standardization: Due to the fact that brightness and contrast vary between raw patch-samples, it is useful to apply *standardization*. Therefore we follow Coates *et al.* and standardize all patches by subtracting the mean of their intensities. Further, we also divide by the intensity standard deviation. This is very important as it increases the significance and quality of the found dictionary. The following equation shows the standardization term, where $\mathbf{x}_m$ are unstandardized raw patch-vectors and β is an additional constant to avoid dividing by zero:

$$\tilde{\mathbf{x}}_m = \frac{\mathbf{x}_m - \text{mean}(\mathbf{x}_m)}{\sqrt{\text{var}(\mathbf{x}_m) + \beta}} \tag{4.1}$$

Assuming that d is the dimensionality of a patch-vector $\mathbf{x}_m$, the mean() operator computes the mean value $\frac{1}{d}\sum_{i=1}^{d} x_{m,i}$ and the var() operator denotes the variance of a patch $\frac{1}{d}\sum_{i=1}^{d}(x_{m,i} - mean(\mathbf{x}_m))^2$.

As a result, we get a standardized patch-pool $\tilde{\mathbf{X}}$. However, if we apply dictionary learning on this standardized data we still get dictionaries of insufficient quality. For explanation we look at a simple K-means clustering approach. We denote dictionary elements as filters in vector-form $\mathbf{d}^{(k)} \in \mathbb{R}^d$ and define a dictionary $\mathbf{D} \in \mathbb{R}^{K \times d}$. On

natural image patches for instance, Coates *et al.* [22] find a dictionary of filters which are illustrated in Figure 4.3a. As it can be seen, the found centroids represent low-frequency edge-like features. However, Coates *et al.* show that this leads to poor performance, although of canceling out influence of contrast and brightness. An explanation for this are correlations between nearby neighboring pixels and due to this, K-means clustering tends to seek more correlated centroids instead of more evenly distributed ones. A 2D toy example of this problem is shown in Figure 4.3b (Coates *et al.* [22]): On the left we see unwhitened and correlated data, whereas on the right whitening rescales the data and removes correlation [48]. Resulting centroids in Figure 4.3a are based on unwhitened data. In contrast, if dictionary learning uses standardized and whitened data, one obtains sharper filters that describes higher spatial frequencies, as seen in Figure 4.3c.



(a) **(b)** **(c)**

Figure 4.3: How whitening affects dictionary learning using K-means on natural image-patches (Coates *et al.* [22]): **(a)** centroids learned from images-patches without whitening. **(b)** A 2D toy example illustrating the effect of whitening on K-means clustering. Left: unwhitened and more correlated data where found centroids tend to be biased. Right: whitened data, yielding more orthogonal centroids. **(c)** centroids learned from whitened image-patches.

Whitening: To decorrelate patch-data and to equalize all sample variances we apply the Zero-phase Component Analysis (ZCA) [6], also called ZCA whitening. Therefore we use the standardized patches $\tilde{\mathbf{x}}_m \in \mathbb{R}^d$ in $\tilde{\mathbf{X}} \in \mathbb{R}^{M \times d}$ and perform the eigendecomposition on its covariance matrix $\mathbf{C} = \tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$ (size $d \times d$), such that $\mathbf{C} = \mathbf{V}\mathbf{E}\mathbf{V}^\top$, where $\mathbf{V} \in \mathbb{R}^{d \times d}$ represents the eigenvectors as column vectors, and $\mathbf{E} \in \mathbb{R}^{d \times d}$ denotes the diagonal matrix of the corresponding eigenvalues. To finally perform ZCA whitening, we apply the transformation matrix $\mathbf{T}_{zca} \in \mathbb{R}^{d \times d}$ according to:

$$\mathbf{T}_{zca} = \mathbf{V}(\mathbf{E} + \mathbf{I}\epsilon_{zca})^{-\frac{1}{2}}\mathbf{V}^\top, \tag{4.2}$$

$$\mathbf{X}_{zca} = \tilde{\mathbf{X}}\ \mathbf{T}_{zca}, \tag{4.3}$$

where $\mathbf{I}$ denotes the identity matrix which is of the same size as $\mathbf{E}$. The constant ϵ_{zca} which is added to the eigenvalues has to be chosen with care. Lower values will amplify higher frequencies as well as undesired noise. However, its optimal value depends on the input data range and has to be found by hand, which denotes a drawback.

$$\mathbf{T}_{pca} = \mathbf{E}^{-\frac{1}{2}}\mathbf{V}^{\top} \tag{4.4}$$

Note that the ZCA whitening is very similar to the PCA whitening which uses the transformation matrix $\mathbf{T}_{pca}$ stated in Equation 4.4. In general, if data is whitened it will stay whitened, even after applying arbitrary data rotation. Comparing both transformation matrices, we see that the only difference is an additional rotation by $\mathbf{V}$ for $\mathbf{T}_{zca}$ (neglecting the factor ϵ_{zca}). Thus, this additional rotation does not affect our resulting dictionary.

The reason why we still prefer ZCA whitening instead of PCA whitening is, that image characteristics are preserved which allows us to visualize learned filters (from dictionary learning) as recognizable images [52]. During experiments, this facilitates examining dictionary learning effects.

As mentioned, after preprocessing, we consider applying two different dictionary learning methods, *sparse coding* and *K-means clustering*, which both assign pseudo-label information to our required patches. For simple notation we henceforth denote the preprocessed patch-pool $\tilde{\mathbf{X}}_{zca}$ as $\tilde{\mathbf{X}}$.

4.1.2 K-means Clustering

At first we want to discuss an optimized version of *K-means clustering* that uses damped centroid updates. The goal is to obtain a dictionary and group patches that share similar characteristics, forming one of K centroids. We also term resulting centroids as the filter kernels of the dictionary. The number of centroids can be arbitrary and as we discuss below, this also controls the final representation dimensionality.

To find a dictionary $\mathbf{D} \in \mathbb{R}^{K\times d}$ of K d-dimensional filters, we randomly initialize centroids based on a Normal-distribution and normalize them to unit length. This fits the scale of the already preprocessed input data $\tilde{\mathbf{X}} \in \mathbb{R}^{M\times d}$ and allows to find more orthogonal centroids [22]. The main objective of this modified K-means is to minimize the squared distance between an input patch $\tilde{\mathbf{x}}_m$ and its reconstruction $\mathbf{s}_m\mathbf{D}^{\top}$, with respect to the dictionary $\mathbf{D}$ and the *code-vector* $\mathbf{s}_m \in \mathbb{R}^{1\times K}$:

$$\min_{\mathbf{D},\mathbf{s}} \sum_m \left\| \mathbf{s}_m\mathbf{D}^{\top} - \tilde{\mathbf{x}}_m \right\|_2^2 \tag{4.5}$$

Hence, the emerging code-vector represents the input patch-vector according to the centroids in $\mathbf{D}$. However, the optimization has to satisfy two constraints: first, each $\mathbf{s}_m$ is forced to have at most one non-zero entry by

$$\|\mathbf{s}_m\|_0 \leq 1, \forall m, \tag{4.6}$$

and second, each centroid $\mathbf{d}^{(k)} \in \mathbb{R}^{1\times d}$ within a dictionary has to have unit length

$$\left\| \mathbf{d}^{(k)} \right\|_2 = 1, \forall k. \tag{4.7}$$

Algorithm 1, stated below, shows the standardized K-means Dictionary Learning, following the works of Coates *et al.* [20, 22]. Note that $\mathbf{S} \in \mathbb{R}^{M \times K}$ represents all code-vectors as a matrix. Applying this algorithm yields very sparse code-vectors $\mathbf{s}_m$ that represent a patch

Algorithm 1 Standardized K-means Dictionary Learning

Require: Dictionary size K
Require: Preprocessed patch-pool $\tilde{\mathbf{X}} \in \mathbb{R}^{M \times d}$
Require: Randomly initialized dictionary $\mathbf{D} \in \mathbb{R}^{K \times d}$
1: **while** not converged **do**
2:

$$s_m^{(k)} = \begin{cases} \mathbf{d}^{(k)} \tilde{\mathbf{x}}_m^\top & \text{if } k = \text{argmax}_l \, |\mathbf{d}^{(l)} \tilde{\mathbf{x}}_m^\top|, \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

3: $\quad \mathbf{D} \leftarrow \mathbf{S}^\top \tilde{\mathbf{X}} + \mathbf{D}$
4: $\quad \mathbf{d}^{(k)} \leftarrow \mathbf{d}^{(k)} / \left\| \mathbf{d}^{(k)} \right\|_2 \ \forall k$
5: **end while**
6: $l_m = \text{argmax}_k \ s_m^{(k)} \ \forall m$
7: **return** Dictionary $\mathbf{D}$ and labels $\mathbf{l}$

by only indicating the distance to the closest centroid (Equation 4.8). With respect to general clustering approaches this is also known as hard-assignment.

Before we explain how to use these code-vectors for RF training, the next section addresses sparse coding which is similar to K-means.

4.1.3 Sparse Coding

Compared to K-means based dictionary learning, *sparse coding* [67] optimizes the same type of objective but uses a different constraint for the complexity of our code-vector $\mathbf{s}_m \in \mathbb{R}^K$. The L1-penalized sparse coding formulation,

$$\min_{\mathbf{D},\mathbf{s}} \sum_m \|\mathbf{s}_m \mathbf{D} - \tilde{\mathbf{x}}_m\|_2^2 + \lambda \|\mathbf{s}_m\|_1 \tag{4.9}$$

considers an additional penalty term $\lambda \|\mathbf{s}_m\|_1$ which forces the code vector to be sparse according to λ. Again, a row-vector of $\mathbf{D}$ is represented as filter $\mathbf{d}^{(k)}$ and is normalized to unit length:

$$\left\| \mathbf{d}^{(k)} \right\|_2 = 1, \forall k \tag{4.10}$$

In contrast to the K-means constraint ($\|\mathbf{s}_m\|_0 \leq 1, \forall m$), the code-vector $\mathbf{s}_m$ is now allowed to have more than one non-zero entry which includes positive and negative values. This provides a more accurate reconstruction of $\tilde{\mathbf{x}}_m$ while still ensuring sparsity. The drawback of this method is that we solve the objective using the coordinate descent algorithm [81], which requires way more training time than K-means clustering. Due to this, we primarily use K-means for dictionary learning in our experiments, as it allows

to learn larger dictionaries and thus larger code-vectors without massively increasing processing time.

Now that we have learned patch code-vectors or patch representations, we exploit this pseudo-label data and train a RF.

4.2 Training a Random Forest as Feature Extractor

As discussed in previous sections we employ two different dictionary learning (DL) approaches to obtain patch code-vectors. The next step is to train the RF as a feature extractor. For this we assume code-vectors to be assigned to *raw* image patches which we use as training data. This yields a RF which is applicable to encode raw input data.

We define $\mathcal{X} = \{\mathbf{x}_m, \mathbf{v}_m\}_{m=1}^{M}$ as our training-set. Further, the characteristics of corresponding pseudo-label vectors $\mathbf{v}_m \in \mathbb{R}^K$ depend on the underlying DL: If we apply K-means clustering (KM) a code-vector $\mathbf{s}_m^{(KM)}$ with one non-zero entry emerges. This value describes the distance to the closest dictionary element (filter) and represents a kind of voting (see Equation 4.8). However, this weighting can be neglected as each patch only belongs to one dictionary element. Hence, we use a so-called "one-hot encoding" scheme for our patch pseudo-label vector and thus assign a 1 at the position of the non-zero entry in $\mathbf{s}_m^{(KM)}$:

$$v_{m,k} = \begin{cases} 1 & \text{if } s_{m,k} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

where $k \in \{1, \ldots, K\}$ is the (pseudo-class) index.

In contrast, for DL with Sparse Coding (SC) we obtain a code-vector $\mathbf{s}_m^{(SC)}$ that is sparsely populated with positive or negative values. Due to this, the code-vector votes for more than one dictionary element. Thus we assume it as pseudo-label vector $\mathbf{v}_m$ without modification.

To train a RF we follow the procedure introduced in Section 3.2 and perform node-sub-sampling [71] with 100 samples. Per node we evaluate $d/2$ split-functions with $n_\tau = 5$ thresholds, where d is the input patch dimensionality. Further we apply Bagging [14] where a bootstrapped subset is of the same size as the training-set but is sampled with replacement. Note that this configuration is found empirically where we accept a tradeoff between efficiency and accuracy.

At next we state how *post-processing* of patch encodings can improve the discriminative power of the final image representation [21]. Hence, we introduce the term post-processing which includes a feature *activation* function and *spatial pooling*. Recall that a patch-encoding is also denoted as feature vector.

Activation of Patch Encodings: Ideally, the trained RF is supposed to produce the same encodings of raw patches as it would be the case if we directly apply the learned dictionary on preprocessed patches, which would be the same as convolutional feature extraction. Depending on the underlying dictionary learning method, which is either KM or SC, we thus get different RF-based features. To enhance the discriminative power of final image representations we may follow Coates *et al.* and additionally "activate" patch-encodings which especially improves the performance of single-layer systems [20–22]. Note that the choice of a proper activation function depends on the global dataset as well as on the application.

Assuming that $\Phi(\mathbf{x})$ encodes an input-patch $\mathbf{x}$ to a feature vector, where $\Phi()$ stands for the mapping function of our trained RF, we consider using three different modes to activate features: (i) we directly use the raw RF-output $\Phi(\mathbf{x})$ as patch encoding $\mathbf{z}$ without using an activation, (ii) we apply soft-thresholding by using the mean-value of $\Phi(\mathbf{x})$ or (iii) we split the positive and negative components of $\Phi(\mathbf{x})$ into separated features (polarity-splitting) which doubles the size of the patch representation $\mathbf{z}$:

$$\mathbf{z} := \begin{cases} \Phi(\mathbf{x}) & \text{(i) no post-encoding} \\ \max(\Phi(\mathbf{x}) - \text{mean}(\Phi(\mathbf{x})), 0) & \text{(ii) soft-threshold activation (mean)} \\ \{\max(0, \Phi(\mathbf{x})), -\min(0, \Phi(\mathbf{x}))\} & \text{(iii) polarity splitting} \end{cases} \tag{4.12}$$

In case of polarity splitting (iii), the following classifier is allowed to weight negative and positive values differently. This has shown to improve performance for sparse features [21]. Soft-threshold activation (ii), however, only passes features voting above average, which induces a slight competition between features. In our experiments we choose different modes for different datasets which we determine empirically, e.g. it turned out that soft-thresholding especially improves performance on rgb-patches which is equal to the findings of Coates *et al.*

To recap, the choice whether an additional activation is useful or not depends on the application and the characteristics of the raw RF-output (influence of the underlying dictionary learning). Further, we also apply spatial pooling on the (activated) feature channels.

4.3 Experiments and Results

In all experiments in this section the goal is to investigate the quality of RF-based image representations by applying image classification. Therefore we apply our RF on raw images, including the test-set and training-set of a given image dataset. Hence, we represent an input image $\mathbf{I}_n \in \mathbb{R}^{H \times W \times q}$ in feature-space $\mathbf{f}_n \in \mathbb{R}^F$, where we refer to the definitions

stated in Equations 2.1 and 2.2. If all images are represented, we perform image classification and train a classifier model on the labeled training data, which is a supervised learning process. Further, we examine the classification accuracy or classification error on the unseen test-set which enables us to evaluate the discriminative power of our extracted representations. Note, as we evaluate random processes we perform several runs (number of runs varies for individual experiments) and track the mean performance.

Overview: The first experiment in Section 4.3.1 shows a proof of concept of our feature extraction framework by classifying gray-valued MNIST-10 images [56]. We compare the representation quality of our RF-based features to others, like those of Coates *et al.* [20, 22], HOG features [31] or concatenated raw image pixel values with image gradient maps.

We further investigate different RF-parameters or pipeline-parameters and show their impact on the resulting representation quality in Section 4.3.2, again using MNIST-10 images. We also evaluate and discuss differences of both dictionary learning approaches, Sparse Coding and K-means clustering. Due to the many pipeline hyper-parameters, we additionally apply a meta-learning approach in Section 4.3.4.

In Section 4.3.3 we try to replicate the results of Coates *et al.* [20] with our own framework and evaluate parameters on the more challenging CIFAR-10 dataset [52]. Hence, we directly compare our feature extraction to the method of Coates *et al.* which both use the same learned dictionary.

Additional experiments in Section 4.3.5 examine how different training-set sizes affect the classification performance. This includes training our RF as well as training a global classifier. Therefore we use the STL-10 dataset [20].

Finally we also add our RF-based representations to an existing image classification pipeline, which was introduced by the VLFeat team [77]. Experiments are shown for CALTECH-101 images [30] and SCENE-67 images [69].

Fixed Parameters: For reasons of simplicity, we fix classifier hyper-parameters for all our experiments. For the linear L2-SVM we set the regularization parameter $\lambda = 100$, as it is proposed in [22]. If not otherwise stated, our non-linear adaBoost classifier is built out of 256 boosted decision trees, using a maximum tree-depth of 4.

Regarding RF-based representation learning on patch-level we consider using two different split-functions. For MNIST images we perform single-pixel tests only, as images are already provided standardized (they are gray-valued without brightness variance). For any other datasets with rgb-images we use pixel-pair tests to capture within-patch characteristics.

4.3.1 Proof of Concept on MNIST-10

The MNIST-10 dataset [56] was introduced by Yann LeCun and Corinna Cortes (Courant Institute, NYU) and consists out of 70000 handwritten digits in total, where each image is gray-valued at a size of 28-by-28 pixels with range 0 to 255. This traditional benchmark dataset is widely used for image classification evaluation and ranks among the "simpler" datasets, as it requires comparably less memory. However, fairly high intraclass variance due to rotations or affine distortions of digits, is still challenging. The dataset comes with a train-set $\mathcal{I}_{train}$ of 60000 images, and a test-set $\mathcal{I}_{test}$ including 10000 images. Because the MNIST-dataset describes decimal digits, the label-space lies within 0 to 9, which defines $C = 10$ global ground-truth categories. Figure 4.4 shows some random examples of MNIST-10 images.

Figure 4.4: Examples of MNIST images [56].

To prove the feature extraction capability of a RF, we first examine a baseline experiment. Table 4.1 shows baseline training parameters of our RF as well as post-processing parameters. Note that post-processing parameters were found empirically. Further, we address unsupervised representation learning with RFs on MNIST-10 and compare our representations to existing ones.

Representation Learning: At first, we build a pool-of-patches of size $M = 10^6$ by randomly sampling over all training images $(\mathbf{I}_n) \in \mathcal{I}_{train}$[1]. For this we just use plain gray-valued image pixels and rescale them to range $[0, 1]$. After patch standardization with $\epsilon_{ZCA} = 1, \beta = 1$, we obtain patch labels by applying K-means clustering [20] (running 10 iterations) where we learn $K = 36$ filter kernels. Note that this small dictionary size is

[1]Note, due to unsupervised training we neglect the provided ground-truth labels c_n.

baseline RF parameters	
patch-size	6-by-6
patch-pool size	$M = 10^6$
number of trees	$T = 20$
maximum tree depth	16
minimum leafnode-samples	100
post processing	
extraction stride	1
encoding activation	none
pooling grid-size	4-by-4
pooling operation	max

Table 4.1: Baseline setting for representation learning on MNIST.

contrary to the findings of Coates *et al.* [20] who achieve best results with largest dictionaries, however, in this experiment we therefore obtain lower dimensional representations that allow fair comparison to other types of features with similar length.

Examples of learned filters using MNIST patches are shown in Figure 4.5. As seen, we get edge-like filters at certain spatial frequencies and different orientations. Notice, that due to the small dictionary size $K = 36$ the algorithm is not really able to output more complex filters that may group curved edge-like appearances. According to Table 4.1 we train a RF and follow the training procedure described in Section 4.2. Because MNIST images already provide standardized data (all images have equal data range) it is sufficient to use the simplest kind of split-function, the single-pixel test ($\Gamma(\mathbf{x}) = x_d$).

Feature Extraction: To produce an image representation we apply our trained RF in a pixel-wise manner by sliding over an input image $\mathbf{I}_n \in \mathbb{R}^{28\times28}$ using a window size of 6-by-6 pixels. This results in a feature-cube with 36 feature channels, each of spatial size 23-by-23. In regard to the found filters shown in Figure 4.5, we illustrate corresponding RF responses for four different input digit images in Figure 4.6. We observe that most of the feature channels describe edge-like information, shifted in various ways at different orientations, further, also response to homogeneous regions appears. As expected, this result underlies the characteristics of the found filters during dictionary learning.

Representation Comparison: To evaluate the representational power we perform image classification. We track 10 runs where for each run we again extract random patches and learn a new dictionary. Hence, this also yields a new RF for each run.

After encoding each image-patch, we apply spatial max-pooling over a grid of size 4-by-4 which captures cells of size 6-by-6. Thus, we obtain an image representation $\mathbf{f}$ which spans $4 \times 4 \times 36 = 576$ dimensions.

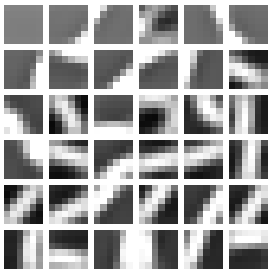


Figure 4.5: Visualization of a found dictionary of size $K = 36$ on MNIST patches using K-means clustering [20]; filters are sorted by filter-variance.

As mentioned we aim at comparing the discriminative power of our representations to others. Therefore we evaluate four different types of feature representations $(\mathbf{f}_n^{(i)}, c_n)$ where i denotes the different feature data. We evaluate the same classification pipeline for each of them.

The feature extraction method of Coates *et al.* [20] directly uses the found dictionary to obtain representations. Hence, to produce patch representations we simply compute the inner product between dictionary filters and preprocessed image-patches, which actually is a convolutional feature extraction. Note that their method requires standardized and whitened input data, because this suits the scale of the standardized filters obtained by the optimized K-means clustering. Resulting patch-encodings are then activated by soft-thresholding using the patch mean-value (thus they name it "K-means triangle-activation"). To allow fair comparison, we assume the same pooling parameters as well as the same found dictionary to train our RF. Equal to our approach, this yields image representations of length 576.

Further we employ low-level image characteristics as feature information. Therefore we consider concatenating three types of feature-channels: (i) raw pixel values, (ii) the gradient magnitudes of the 2D image and (iii) eight oriented gradient channels, where we use the binning $\{0, \frac{\pi}{8}, \frac{\pi}{4}, \frac{3\pi}{8}, \frac{\pi}{2}, \frac{5\pi}{8}, \frac{3\pi}{4}, \frac{7\pi}{8}\}$. This results in 10 feature channels of size 28×28, which we compute using the toolbox of Piotr Dollar[2][28] for MATLAB. For fair comparison we apply spatial pooling over a pooling-grid of 8-by-8, where sum-pooling turned out to work best. Resulting features are stacked to one representation vector, spanning $8 \times 8 \times 10 = 640$ dimensions.

Finally, we also test a widely used hand-crafted feature-descriptor, the HOG-descriptor. Again, we use the toolbox of Piotr Dollar [28] which provides the `fhog` function that efficiently computes HOG-features based on the work of Felzenszwalb *et al.* [31]. We set a HOG cell-size of 4-by-4 and compute 9 gradient orientations. It internally focuses on

[2]Piotr's Image & Video toolbox 3.23 ©2013 Piotr Dollar

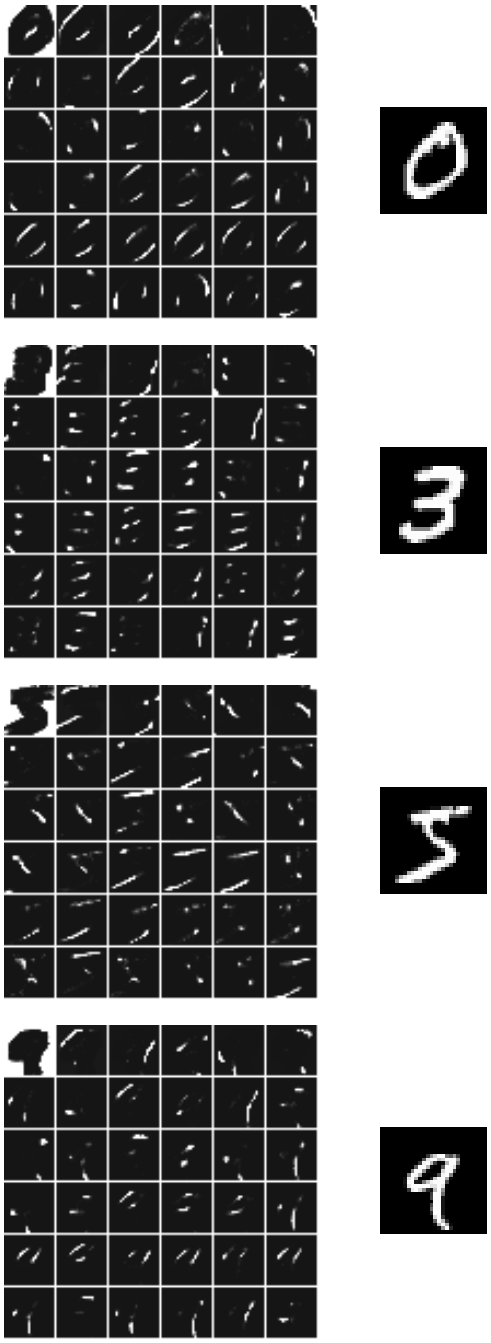


Figure 4.6: An illustration of $K = 36$ RF-based feature channels for several MNIST digits: From top to bottom we show resulting feature channels for digit "0","3","5" and "9". To the right, the corresponding input images are shown.

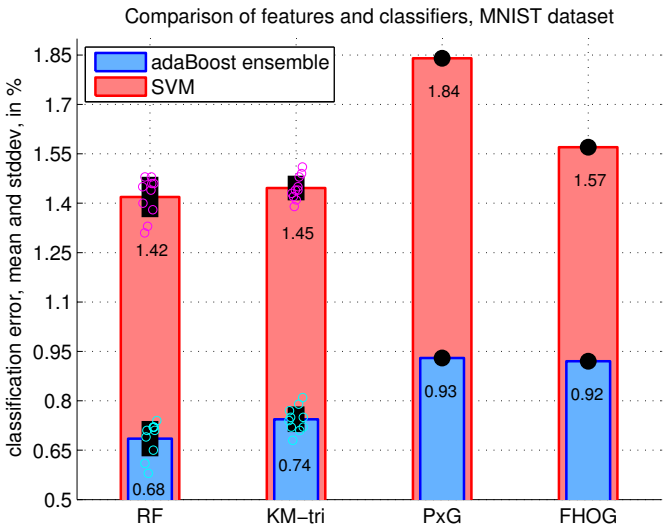


Figure 4.7: Image classification results on MNIST, a comparison of representations. `RF` denotes RF-based image representations; `KM-tri` denotes triangle K-means representations [20]; `PxG` states a concatenation of raw pixels, gradient magnitudes and oriented gradients; `FHOG` describes histograms of oriented gradients. Markers show classification errors of individual runs.

mean error in %	feature dimensions	SVM	adaBoost ensemble
`pxl+gradM+gradH`	640	1.840	0.930
`FHOG`	512	1.5700	0.920
`K-means tri` [20]	576	1.446 ± 0.037	0.744 ± 0.038
`RF`	576	$\mathbf{1.419} \pm 0.061$	$\mathbf{0.685} \pm 0.054$

Table 4.2: Image classification results on MNIST, a comparison of representations. Mean classification error and standard deviation in %.

2 contrast-sensitive features and one contrast-insensitive feature for each orientation. In addition to that, also four texture features are computed. As a result we get 32 features for each cell that are clipped at a value of 0.2. Thus, an image representation is of length 512.

If we look at the classification results in Figure 4.7 and in Table 4.2, we see that our RF representations outperforms all other tested representations. Notice, although the pseudo-label information of the patches is based on the same learned dictionary as for the triangle K-means, our representations seem to be more discriminative. An additional benefit is that the RF directly encodes raw patch-data *without* any preprocessing of input image patches. In contrast to the findings of Coates *et al.* who achieve best results for very large dictionaries, we apply a very small dictionary size ($K = 36$) in this experiment. The reason for this is that we focus on producing similar image representation lengths for all four methods, allowing a fair comparison.

The next experiments evaluate our RF-based image representations using larger dictionaries where we further investigate the influence of RF parameters or representation learning parameters.

4.3.2 Experiments on MNIST-10 - Evaluation of Parameters

The goal of the following experiments is to examine the impact of different parameters on the final representation quality. In the first part we investigate RF related parameters whereas in the second part we examine the influence of different dictionary sizes and patch-sizes. Further we compare two unsupervised dictionary learning methods, K-means (KM) and sparse coding (SC). We again use the MNIST-10 dataset and perform image classification using the non-linear adaBoost classifier, as it showed best classification rates in our previous proof-of-concept experiment (on MNIST data). If not otherwise stated we use the MNIST baseline parameters stated in Table 4.1.

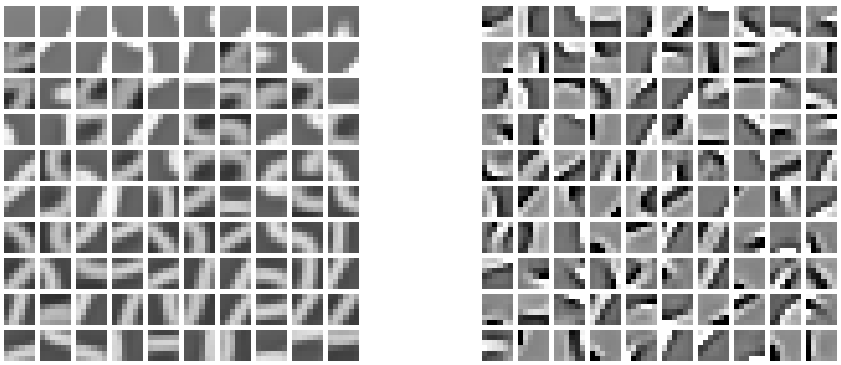
Evaluation of RF Parameters: At first we examine the influence of RF parameters on image classification. To learn features we apply KM clustering or SC and use a fixed dictionary size of $K = 100$ for both methods. However, empirical trials showed that best results are achieved by using the raw RF output for KM dictionary learning, and the polarity splitting activation if dictionaries are learned with SC (for more detail the reader is referred to Section 4.2). Note that if we apply polarity splitting, our final representation length is of doubled size. Thus, with a pooling-grid of size 4-by-4, we get $4 \times 4 \times K$ features for KM and $4 \times 4 \times 2K$ for SC.

Further we fix the following dictionary learning hyper-parameters: For preprocessing we use $\epsilon_{ZCA} = 1, \beta = 1$ for KM, and $\epsilon_{ZCA} = 0.1, \beta = 1$ for SC, which were empirically found to yield best performances. Both dictionary learning methods perform 10 optimization iterations, where the SC L1-penalization parameter λ is set to 1.

Examples of learned dictionaries for both algorithms are illustrated in Figure 4.8 where it can be seen that resulting dictionary filter kernels look quite different. We evaluate the RF **max-depth** $\in \{1, 4, 7, 10, 13, 16, 20, 30, \infty\}$ and vary the total **number of** trained **decision trees** for each max-depth using $T \in \{1, 3, 10, 32, 100\}$. For training our RF, we additionally consider using either 100 or 10 minimum leafnode samples (mls) that contribute to the terminal leafnode statistics.

Figures 4.9a to 4.9e show achieved results for each RF size T, drawn over the maximum allowed tree-depth. As performance measurement we show the mean classification-error over 5 runs per parameter-setting.

As expected, if we increase the number of trees the ensemble prediction strength of our RF improves and obviously yields better features. Notice, that also a single tree provides surprisingly well feature extraction where we get a minimum mean-error rate of 0.726% using a maximum tree depth of 16 with SC-based feature learning.



(a) **(b)**

Figure 4.8: Visualization of found dictionaries on 6-by-6 MNIST patches, using $K = 100$. **(a)** dictionary learned by K-mean; **(b)** dictionary learned by sparse coding; both sorted by filter-variance.

If we examine results using RFs with 100 mls, we see increasing classification performance up to a max-depth of 16 for nearly all ensemble sizes. However, beyond this max-depth it seems that results show fluctuations, where we observe a more or less constant standard deviation of 0.04% for almost all errors. Hence, we conclude that beyond this point representation quality does not further improve, although more pixel-tests are performed on image patches. Even more, some mean errors increase with max-depths beyond 16 which can be explained by slight overfitting.

Looking at the results which use RFs with 10 mls, we recognize higher overfitting characteristics, in particular for fewer trees. This is reasonable as if we only train e.g. one decision tree, ensemble advantages stay out. Regarding fewer mls, decision trees are enabled to perform more splits, resulting in a significantly higher number of final leafnodes, which contributes to overfitting. In this case, limited memory did not allow us to train deeper RFs. As expected, a higher number of trees shows to yield better generalization which counteracts overfitting especially for 10 mls.

Regarding experiments with 100 mls and max-depth $= \infty$, we did not exceed memory as depth was limited by the minimum number of leafnode samples which turned out to allow tree depths of about 50.

If we use ensembles of simple stumps (depth = 1), we achieve mean-error rates below 1 % for $T \geq 32$. Hence, our experiments show that it suffices to apply at least 32 binary decisions on each raw image patch to obtain representations for acceptable image classification on MNIST.

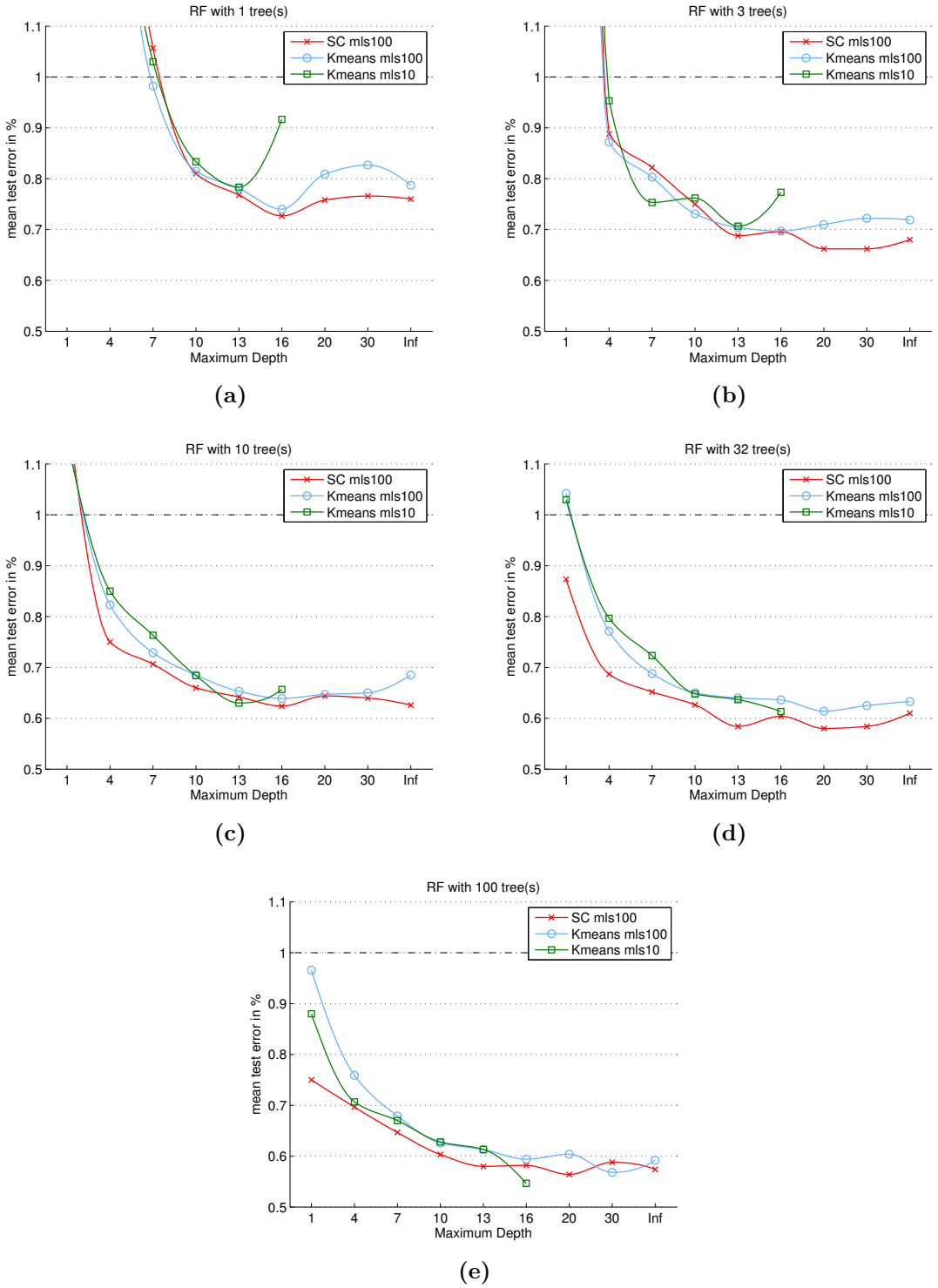


Figure 4.9: RF parameter sweep of maximum tree-depth and number of trees T. Figures **(a)** to **(e)** show mean classification errors over 5 runs for different number of trees. The rough standard deviation for almost all settings is 0.04%. During RF training we use either 100 or 10 minimum leafnode samples (mls) for computing leafnode statistics. SC and Kmeans denote the dictionary learning methods, sparse coding and K-means clustering, both learning $K = 100$ filters.

Comparing the two dictionary learning approaches, we see that SC facilitates to extract slightly better RF-based representations. However, remember that we additionally have to apply an activation (polarity splitting) on the raw RF-output. Otherwise we empirically found that performance decreases. Also note that SC is way slower than KM, where we refer to Section 4.1.3 for further information.

The best results in this parameter sweep showed a minimum mean classification error of 0.54% (mls50) and 0.57% (mls100) for KM dictionary learning, and 0.56% (mls100) for SC dictionary learning, all using a RF size of $T = 100$. As expected, using different minimum leafnode samples has a significant impact on the final number of RF leafnodes. If we allow at least 50 leafnode samples, RFs with 100 trees and max-depth of 16 learn about 1.4 million leafnodes in total whereas with 100 mls, RFs learn 0.9 million leafnodes. Hence, there is a big difference with respect to memory consumption, although we only achieve a slight improvement for 50 mls.

If we compare above results to the proof-of-concept experiment in Table 4.2 (which only involved dictionaries of size $K = 36$), we see that K does, of course, affect the discriminative power of extracted image representations. Therefore we are further interested in examining the impact of the dictionary size and of the input patch-size on the final image classification performance.

Evaluation of Representation Learning Parameters: In this part we fix the RF-parameters according to the baseline settings ($T = 20$ at depth 16) and evaluate different **patch-sizes** $w \in \{2, 4, 6, 8, 10, 12\}$, whereas for each patch-size we vary the **dictionary size** $K \in \{10, 32, 100, 316, 512\}$. Note: Due to limited computational resources we do not investigate larger dictionaries (e.g. Coates *et al.* [20, 22] showed best performances for $K = 4000$). Again we employ KM and SC for dictionary learning, however, in contrast to KM the SC optimization algorithm restricts complexity in dictionary learning, preventing us from learning large dictionaries due to extremely long runtimes. For each parameter-setting we evaluate 3 runs and examine the mean classification-error.

In Figure 4.10 we illustrate examples of found dictionaries at different patch-sizes for $K = 100$ filters, regarding KM clustering and SC, respectively. We observe that larger filters capture details of larger scale, like curves or even circular shapes, whereas small filters have limited representational power. Classification results for different patch-sizes are shown in the following Figures 4.11a to 4.11f.

As expected, if we increase the dictionary size K we basically improve image classification, except for representations based on small patch-sizes. In these cases

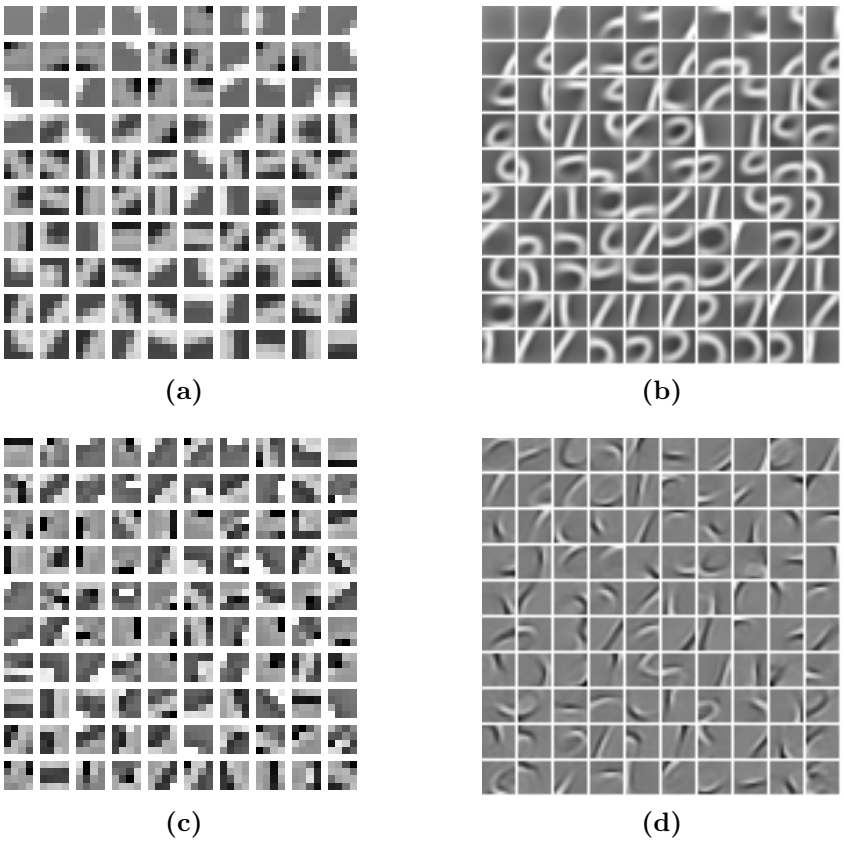


Figure 4.10: Visualization of found dictionaries ($K = 100$) using different patch-sizes of MNIST images. **(a)** KM-based filters at size 4-by-4; **(b)** KM-based filters at size 12-by-12; **(c)** SC-based filters at size 4-by-4; **(d)** SC-based filters at size 12-by-12. Dictionaries are sorted by filter-variance.

we even observe error increase for $K > 100$. This is explainable by the limited representational power of smaller filters for large dictionaries, which is also the reason why SC optimization failed for a quite small patch-size of 2-by-2.

However, if we set $K = 10$, we still achieve surprisingly low classification errors, especially if we apply SC for dictionary learning. Hence we confirm the findings of Coates *et al.* in [21] that show superior behavior of SC over KM with respect to representational power.

Regarding the effect of the patch-size we consider a size of 6-by-6 as the most suitable for MNIST image feature extraction, where the lowest mean classification error of about 0.57% emerged at using $K = 512$. Note that we only used 20 trees with a max-depth of 16. Thus we conclude, that in addition to the RF structure, also a proper patch-size and large dictionaries are critical for optimal performances.

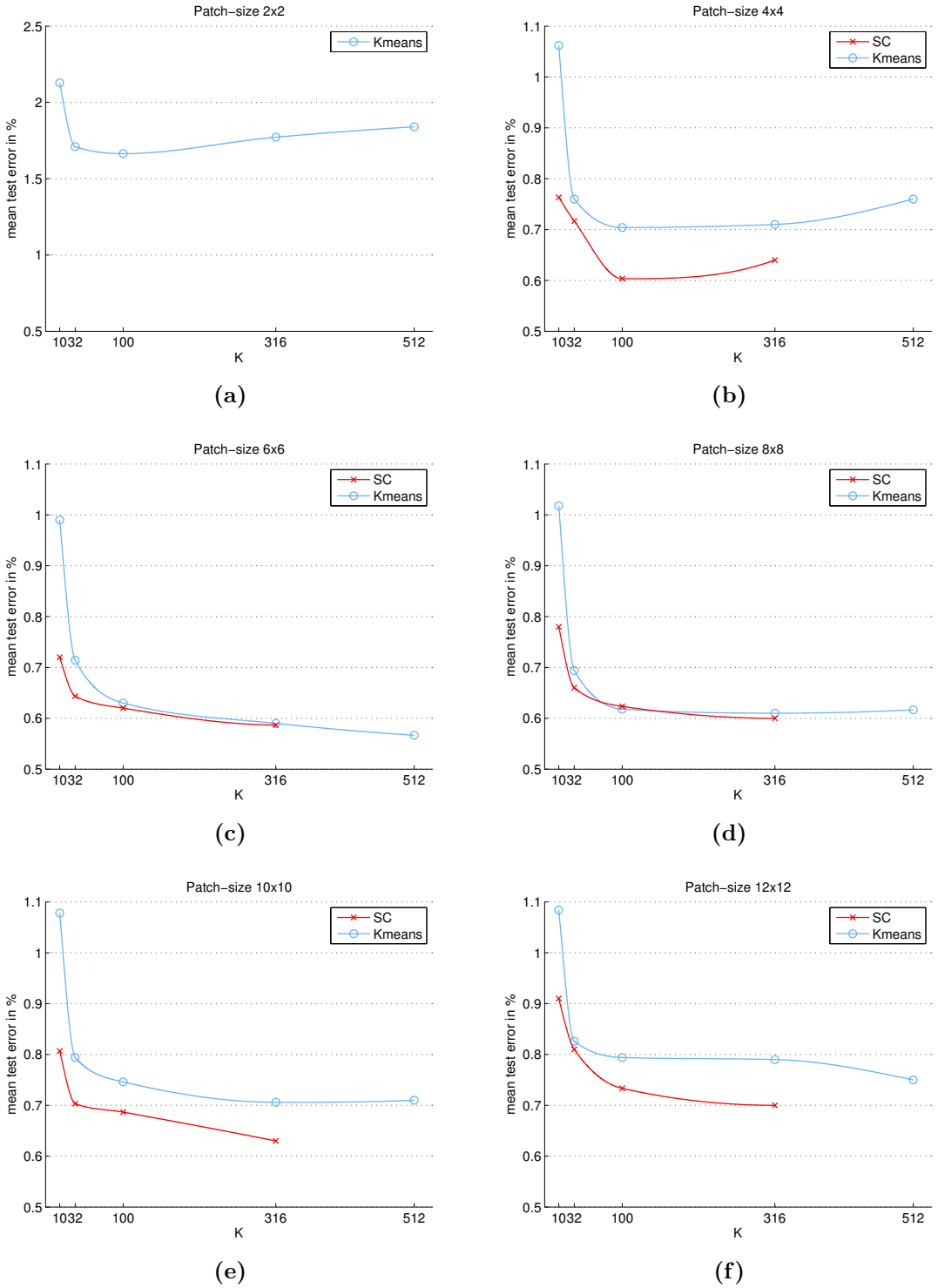


Figure 4.11: Parameter sweep over dictionary size K at different patch-sizes. Figures **(a)** to **(f)** show mean classification errors over 5 runs for different patch-sizes. The rough standard deviation for almost all settings is 0.04%. The plots are drawn over varying dictionary size K with cubic interpolation. SC denotes Sparse Coding. RF parameters are fixed, using $T = 20$ trees at a maximum depth of 16.

Beside extensive research on MNIST images, the next section addresses experiments using more challenging data in terms of real world rgb-images at low resolution.

4.3.3 Experiments on CIFAR-10

In this experiment we perform image classification on CIFAR-10 images [52] and compare performances of our RF-based representations to the proposed method of Coates *et al.* [20, 22]. Further we also investigate feature extraction time efficiency of both methods.

The CIFAR dataset consists out of 60000 rgb-images of size 32-by-32 that show 10 different real-world categories (*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*). Each category has 5000 train-images and 1000 test-images, resulting in 50000 training samples and 10000 test samples in total. The challenge of this dataset are the low-resolution real-world objects which are quite hard to distinguish. As it was found by Karpathy[3], even humans only achieve a recognition accuracy of about 94%. Random examples of the CIFAR-10 images are illustrated in Figure 4.12.

Figure 4.12: Examples of CIFAR images [52].

The main focus of this experiment lies on comparing our representations to the "K-means triangle" (KM-Tri) features of Coates *et.al.*, who primarily evaluate on CIFAR images. Thus, we use the same pipeline-parameter settings for both approaches during evaluation, which includes the type of classifier as well as the same found dictionary by KM. Regarding dictionary learning, we vary the dictionary size $K = \{100, 200, 400, 800\}$ and the whitening parameter $\epsilon_{zca} = \{0.1, 0.01\}$. We also investigate the effect of two different RF architectures and split-functions.

[3]Karpathy, Andrej. *Lessons Learned from Manually Classifying CIFAR-10.* 2011. `http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/` (accessed October, 2016)

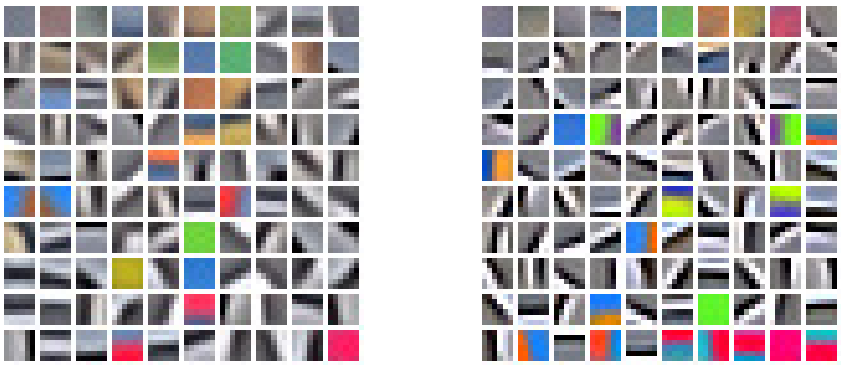
Parameters	RF	DRF
patch-size	6-by-6	6-by-6
patch-pool size	$M = 10^6$	$M = 3 \cdot 10^6$
number of trees	$T = 50$	$T = 10$
maximum tree depth	16	∞
minimum leafnode-samples	100	1000
split-function	single pixel, pixel pair	pixel pair
preprocessing	raw, standardized	raw
post processing		
extraction stride	1	
encoding activation	threshold	
pooling grid-size	2-by-2	
pooling operation	sum	

Table 4.3: Parameter settings of RF-based feature extraction on CIFAR images.

As seen in Table 4.3, we employ a typical RF with 50 trees at max-depth 16, and a deep RF (DRF) with 10 trees where final tree depth is only limited by the minimum number of leafnode-samples (mls). We set mls = 1000 as this allows us to train the DRF on more image-patches, however, this also causes learning quite unbalanced trees which is the reason why we choose infinite depth. As a result, tree depths around 75 appear and approximately 70000 total leafnodes are learned. Notice, that in contrast to our feature extraction on MNIST, we additionally apply a threshold encoding activation which we empirically found to yield best results. Further, this encoding activation is also used during the KM-Tri feature extraction.

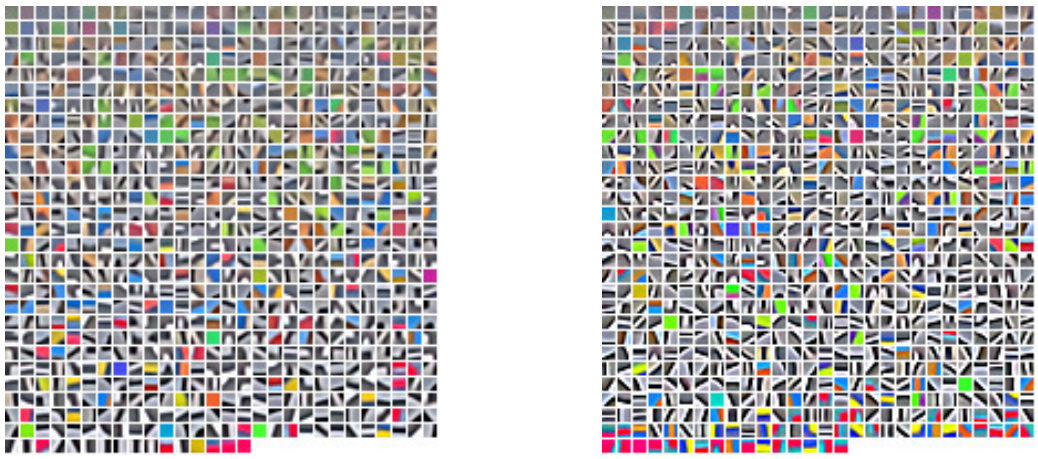
We randomly sample raw 6-by-6 image patches and rescale their range from $[0, 255]$ to $[0, 1]$. After patch preprocessing, we apply K-means clustering with 10 iterations to learn a dictionary and to obtain pseudo-label information. The reason why we omit sparse coding in this experiment is the otherwise too high optimization complexity. Because we have a large number of patches with $\mathbb{R}^{6\times6\times3} = \mathbb{R}^{108}$ dimensions, SC would require quite high processing runtime compared to KM, at least on our machine.

Figures 4.13 and 4.14 show found dictionaries of size 100 or 800, respectively, where we also vary the whitening parameter ϵ_{ZCA}. As seen, larger dictionaries produce more diverse filters, also with respect to color information. Similar to MNIST dictionaries, emerging filters describe edge-like information including different spatial frequencies. Additionally, emerging Gabor-filters vary in scale and some also include mixed colors. As stated in Section 4.1.1, the whitening parameter ϵ_{zca} controls the "sharpness" of dictionary filters which also affects the resulting patch pseudo-label information.



(a) **(b)**

Figure 4.13: Visualization of a learned dictionaries of size 100 using 6-by-6 CIFAR patches. **(a)** Dictionary learning with $\epsilon_{ZCA} = 0.1$; **(b)** Dictionary learning with $\epsilon_{ZCA} = 0.01$, both sorted by filter variance.



(a) **(b)**

Figure 4.14: Visualization of a learned dictionaries of size 800 using 6-by-6 CIFAR patches. **(a)** Dictionary learning with $\epsilon_{ZCA} = 0.1$; **(b)** Dictionary learning with $\epsilon_{ZCA} = 0.01$, both sorted by filter variance.

In this experiment we examine the classification mean accuracy on the test-set over 3 runs. We train two different types of classifiers, a non-linear adaBoost classifier and a linear L2-SVM (for more details see Sections 2.4). In regard to the adaBoost classifier, we consider training with two different settings: We train 256 trees of depth 4 which we denote as setting 1, and we train 512 trees, denoted as setting 2.

Figure 4.15 compares achieved results of both classifiers, using the preprocessing parameters $\epsilon_{ZCA} = \{0.1, 0.01\}, \beta = 1$ for dictionary learning. As seen, our RF-based representations do not outperform KM-Tri representations even though they are based on the same dictionaries. We especially observe this for linear classification. An explanation may

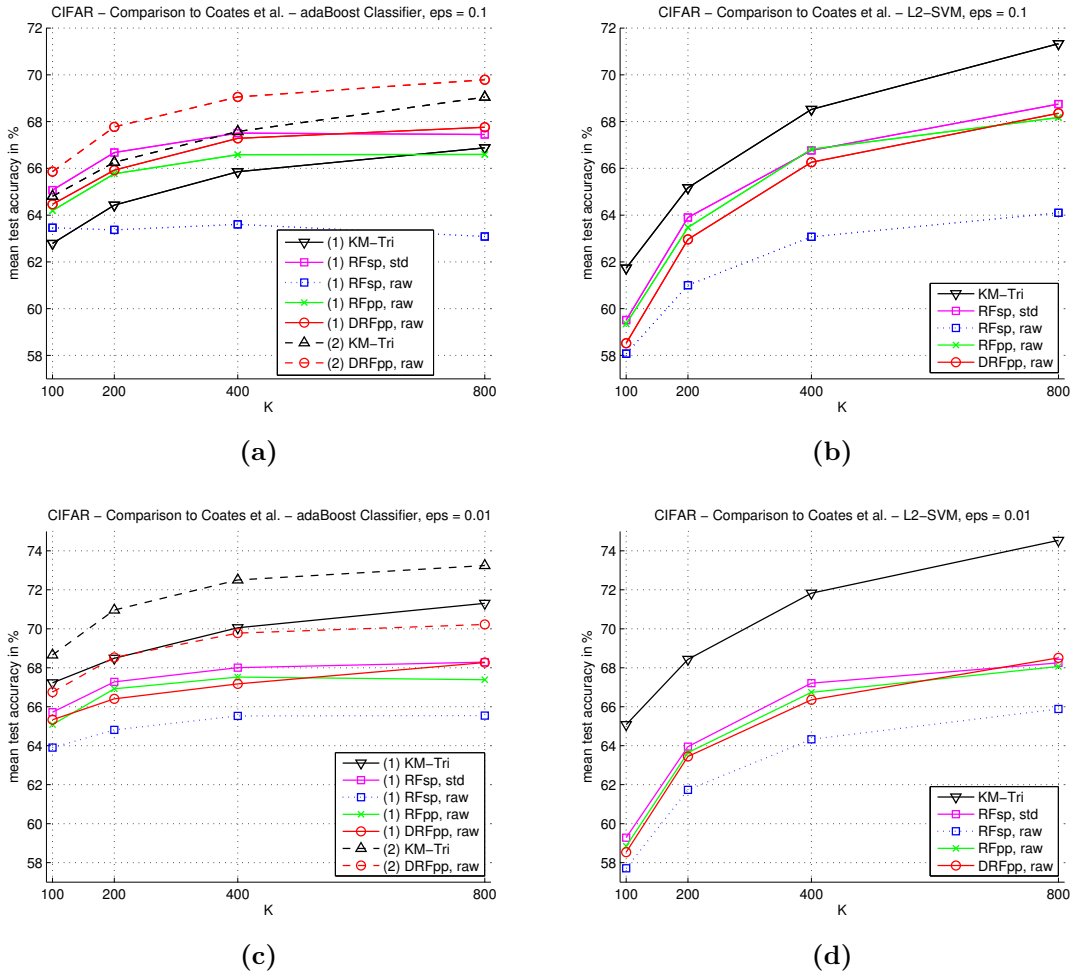
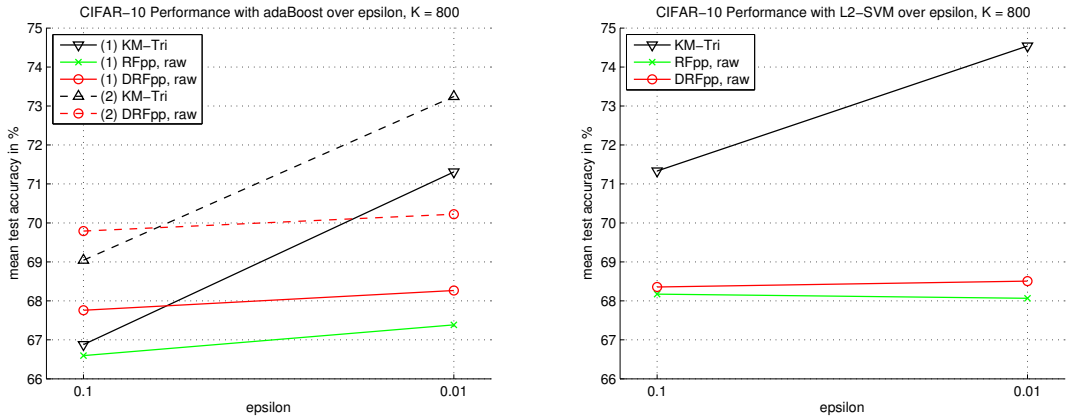


Figure 4.15: Performance comparison over K on CIFAR. For dictionary learning we use $\epsilon_{ZCA} = \{0.1, 0.01\}$ (notice the different scaling). Classification mean accuracy on the test-set over 3 runs is drawn over dictionary size K: (RFsp) RF with single-pixel tests; (RFpp) RF with pixel-pair tests; (DRFpp) Deep RF with pixel-pair tests; (KM-Tri) K-means triangle representations; 'raw' denotes raw input patches, 'std' denotes standardized input patches. Note KM-Tri uses standardized and whitened input patches. **(a)(c)** Results with non-linear adaBoost classification, (1) 256 trees and (2) 512 trees; **(b)(d)** Results with linear L2-SVM classification.

be the well optimized parameters of the entire KM-Tri pipeline that is proposed by Coates *et al.* [20]. However, if we compare performances over different ϵ_{ZCA} we recognize that KM-Tri representations are extremely sensitive to preprocessing parameters, whereas our RF feature extractor shows to be less affected. This can be seen in more detail in Figure 4.16 where we plot the results over the preprocessing parameter ϵ_{ZCA} for $K = 800$. A reason for this is that we do not directly employ the actual dictionary, but the resulting pseudo-label information for RF training, causing our RF-based feature extraction to be more independent on preprocessed dictionary learning. Hence we confirm the findings of



(a) **(b)**

Figure 4.16: Classification performance comparison for different values of the preprocessing parameter ϵ_{ZCA} on CIFAR, using dictionaries of size $K = 800$: (RFpp) RF with pixel-pair tests; (DRFpp) Deep RF with pixel-pair tests; (KM-Tri) K-means triangle representations; 'raw' denotes raw input patches. Note KM-Tri uses standardized and whitened input patches. **(a)** Results with non-linear adaBoost classification, (1) 256 trees and (2) 512 trees; **(b)** Results with linear L2-SVM classification.

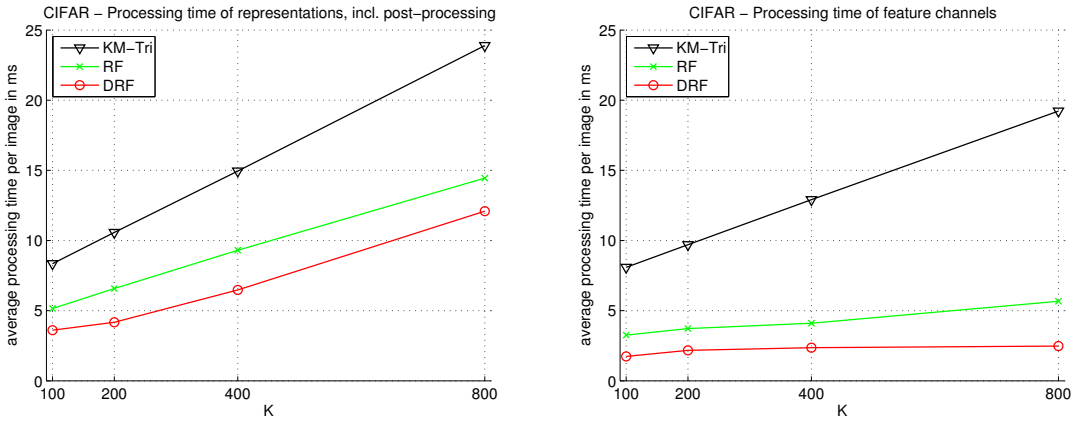
Coates *et al.*, that preprocessing is crucial for their method.

If we take a closer look at the results in Figure 4.15 with non-linear adaBoost classification using 256 trees (setting 1), we do not improve over the linear SVM. However, if our adaBoost forest trains 512 trees (setting 2), performances are increased. This behavior indicates an insufficient model complexity of our non-linear adaBoost classifier which may prevent exhausting the potential of high dimensional image representations.

If we compare the different RF settings, we see that single-pixel tests on standardized input data (RFsp, std) performs quite well. That is, we train and test the RF on standardized patches. However, we primarily aim at extracting features from raw image data. As expected, some patch characteristics, such as edge information, can not be properly learned by our RF if we just use single-pixel tests on raw input patches (RFsp, raw). In this case, two random patches with similar edge-like content may vary in e.g. brightness but causing the RF to output different encodings. Thus we apply pixel-pair split-functions that allow to capture raw patch characteristics (RFpp, raw).

Furthermore we also test a deeper RF, denoted as DRF, but with less trees. Again we train on raw patch data. As it turned out, although we only use 10 deep trees, classification performances are nearly equal to our other RFs who are built out of 50 trees.

Regarding time efficiency, Figure 4.17 shows a comparison of both methods over K. Therefore we use the time measurement tool of MATLAB which unfortunately has flaws if measured time periods are below one second. Thus we evaluate 1000 images and use the



(a) **(b)**

Figure 4.17: Time efficiency comparison of feature extraction methods on CIFAR. Average processing time per image in ms. **(a)** with post-processing, including threshold-encoding and feature pooling; **(b)** without post-processing.

CIFAR-10 classification	SVM		adaBoost, 256 trees		adaBoost, 512 trees	
Dictionary size K	100	800	100	800	100	800
RFsp, std	59.28	68.25	65.72	68.28	-	-
RFsp, raw	57.71	65.88	63.90	65.54	-	-
RFpp, raw	58.84	68.06	65.08	67.38	-	-
DRFpp, raw	58.53	68.50	65.33	68.26	66.75	**70.22**
KM-Tri, std, whitened	65.08	**74.54**	67.22	71.30	68.65	73.24

Table 4.4: Image classification results on CIFAR. Achieved mean accuracies are shown for RF-based representations and KM-Tri representations, using different dictionary sizes.

median time period and regard the average time consumption over 3 runs.

If we examine the extraction time without any post-processing, that is, we only produce image patch encodings or feature channels, we recognize independency on K for RF-based extraction. As expected, extraction time increases with K for KM-Tri. However, in case of including time consumption for threshold encoding and feature pooling, post-processing seems to be fairly time consuming with respect to K. Nevertheless, our method still outperforms convolutional feature extraction regarding processing time.

To recap, best achieved classification performances on CIFAR are shown in Table 4.4. As seen, we did not reach performances of KM-Tri representations which we explain by the extremely well tuned pipeline of Coates *et al.* [20]. Nonetheless we prove, that our feature extraction can be done more efficiently by using raw data, which denotes a major advantage over convolutional extraction. Moreover, our method is less sensitive to

preprocessing during dictionary learning. Note that Coates *et al.* achieve best results with $K = 4000$, however, due to memory limitations we do not test for dictionary sizes beyond 800.

4.3.4 Learning Hyper-Parameters

As we have seen in previous experiments, it is still quite challenging to find optimal parameters or hyper-parameters by hand, regarding a whole classification pipeline. Hence, we are motivated to determine how we could learn them. Therefore we propose to apply a meta-learning approach, to seek an optimal global pipeline parameter setting. However, before we introduce our experiment, we briefly explain evolutionary learning.

Evolutionary learning: Algorithms based on evolutionary learning are used to find parameters or hyper-parameters of global learning problems. A certain parameter-setting is defined as individual within a set of individuals, denoted as population. Further, each individual outputs an error or "fitness" at testing-time which is equal to the performance of the sub-task. Evaluating a whole population may show good or bad individuals where the best performing ones may be chosen to form a new generation. A well-known concept for evolutionary learning are *genetic algorithms* [39]. The key is to evolve individuals (or "genomes") that show best performances on a given problem. Ideally, such algorithms converge to the global optimum which is usually hard to determine with common learning methods. Regarding our experiment, we consider using the following genetic learning approach:

At first, we randomly draw individuals over a pre-defined parameter-space $\mathcal{S}$ to form a population. That is, we produce a random set of pipeline parameter-settings which we denote as the first generation of individuals. Further we evaluate each individual, which means in this context to evaluate an end-to-end classification pipeline on MNIST-10 images. This yields an error or score for each individual. Then, the best ones of this generation are chosen by a predefined criterion to reproduce themselves forming the next generation. During reproduction random mutations may occur which ensures visiting new areas in parameter space. Thus, with advanced generations we get more improved individuals. A pseudo-code explanation of our genetic algorithm we employ in this experiment is stated in Algorithm 2.

We define a parameter-space $\mathcal{S}$ which spans over multiple discrete dimensions that describe all possible discrete states. As mentioned our goal is to learn an optimal constellation or "tuple" of parameters, which denotes an individual. Table 4.5 shows the possible range for pipeline parameters in this experiment.

Given this parameter-space the resulting image representation may span 32 to 10800 dimensions, which only depends on K and the selected pooling size. In fact, the patch-size only affects discriminative characteristics of patch-representations with respect to scale before spatial pooling is applied. For dictionary learning we consider using K-means

Algorithm 2 A Genetic Algorithm to learn parameter-settings

Require: Define parameter-space $\mathcal{S} \in \mathbb{R}^D$ where D = number of genes.[a]
Require: Initialize population uniformly drawn over parameter-space $P_0 \sim U(\mathcal{S})$.
Require: Define probability for mutating an individual p_{ind}.
Require: Define probability for the single gene mutation p_{mut} of an individual.
Require: Set maximum number of generations G.
Evaluate fitness R_0 of initial population P_0.
for $g = 1 \rightarrow G$ **do**
    Select survivors from fitness R_{g-1}: $P_{survivors} \leftarrow \text{select}(P_{g-1})$. [b]
    Crossover individuals until population-size is restored $P_g \leftarrow \text{crossover}(P_{survivors})$.
    for $k = 1 \rightarrow$ population-size **do**
        Consider current individual $I_k \in P_g$ for mutation according to probability p_{ind}.
        if mutate I_k **then**
            Mutate each gene of I_k according to probability p_{mut}: $I_k \leftarrow \text{mutate}(I_k)$.
            Update population P_g.
        end if
    end for
    Evaluate actual generation: $R_g \leftarrow \text{eval}(P_g)$.
    Check if termination criterion is met, otherwise continue.
end for
return best individual I_{best} of last population.

[a] Note that $\mathbb{R}$ is not restricted to continuous spaces here, it may also denote discrete spaces.
[b] For select() we consider using the "elitism" principle, which selects the best n_e individuals to survive.

parameters	**parameter-space $\mathcal{S}$**
# RF trees	$T = \{1, 2, 3, \ldots, 99, 100\}$
maximum RF tree depth	$\{2, 3, 4, \ldots, 9, 10\}$
# of learned filters	$K = \{8, 16, \ldots, 300\}$, step-size 16
patch-size	$w = \{2, 4, \ldots, 14\}$, step-size 2
spatial max-pooling grid	cell-size {2-by-2, 4-by-4, 6-by-6}
K-means whitening parameter	$\epsilon_{ZCA} = \{1, 0.1, 0.01, 0.001\}$
encoding activation	{'raw', 'pn', 'threshold'}

Table 4.5: Defined parameter space $\mathcal{S}$ for parameter learning.

clustering, as it is way faster than Sparse Coding.

Assuming our first generation, the initial population P_0 contains 20 random individuals. The mutation probability, which decides whether an individual mutates, is set to $p_{ind} = 0.3$ and the following single-gene mutation probability is set to $p_{mut} = 0.3$. Following Algorithm 2, we consider choosing the best 10 individuals as survivors and thus for reproduction. We evaluate 30 generations and record the iterative evolutionary progress of the population mean-error and its standard deviation, as well as the minimum error of

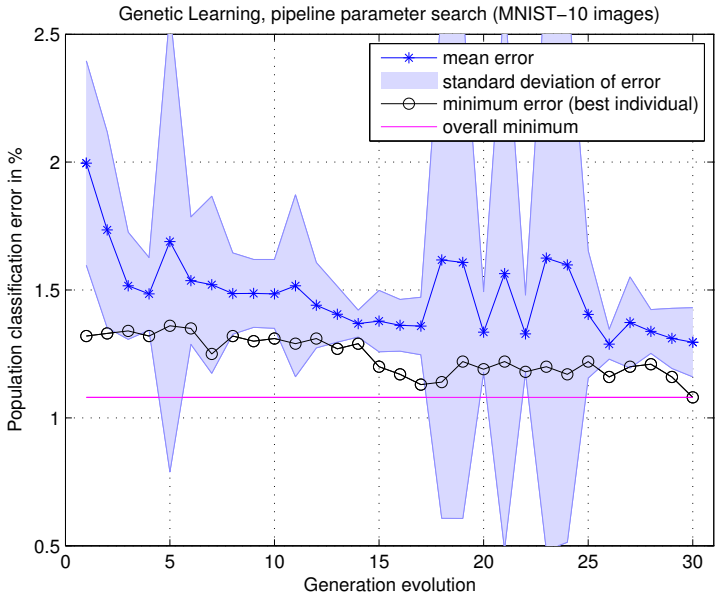


Figure 4.18: Evolution of pipeline performances during a hyper-parameter meta-learning approach, see Algorithm 2. Classification error on 10000 MNIST-10 test images, using 10000 training images only.

each generation (which equals the best individual). If we talk about an error we denote the classification-error on the 10000 test-images. As classifier we use the adaBoost tree ensemble. Note that we only use 10000 training images here, as this reduces the total runtime of our genetic learning algorithm. In Figure 4.18 we plot the evolutionary learning progress over 30 generations.

As seen, the learning curve shows some random fluctuations that are caused by random mutations. Still, generations improve on average during learning. The best population mean-performance occurs at the 26-th generation where we achieve a mean-error of 1.288% over 20 individuals. The best overall *individual* performance emerges in the last generation where we attain an error-rate of 1.13%, whereby the following genes or parameters evolved: $T = 57, \text{max. depth} = 10, K = 264, \text{patch-size} = 6 \times 6, \text{pooling-grid} = 2 \times 2, \epsilon_{ZCA} = 1, \text{RF encoding} = \text{'raw'}$. Notice, in contrast to our previous experiments on MNIST, we get a pooling-grid of size 2-by-2. As expected the algorithm yields a maximum tree depth of 10 which denotes the upper limit of the depth parameter space. Note that we did not allow the algorithm to larger tree depths since this would reasonably increase genetic learning time. Though, deeper trees usually improve performance.

Finally, we train our classification pipeline according to this "optimal" parameter setting but consider using the full training-set with 60000 images. In this case we achieve a mean classification error of 0.62% over 3 runs on the test-set. Further we increase the maximum tree depth to 16 which yields an improvement of the mean error to 0.54%.

To recap, hyper-parameters of an entire image classification pipeline can be learned automatically by applying evolutionary learning. Thus, although we introduce additional hyper-parameters with our RF, tedious parameter tuning by hand could be avoided.

4.3.5 Miscellaneous Experiments

In this section we address additional experiments on image classification using RF-based representations.

Augmenting Training Images for Classification: At first, we investigate the influence of the image training-set size with respect to global image classification. We evaluate on the MNIST-10 dataset which provides $N_{train} = 60000$ training images by default. To augment this dataset we apply affine image transformations on given training images. Hence, we produce new digit images that slightly vary in rotation, shear and scale.

For representation learning we use the MNIST baseline setting stated in Table 4.1, but train $T = 32$ trees at max-depth 10. Note that we also involve augmented training data for representation learning with respect to image-patch sampling. However, we still use 10^6 patches for dictionary learning where we learn $K = 100$ filters. All other parameters are considered to be fixed. The results in Table 4.6 show that data augmentation

N_{train}	$30 \cdot 10^3$	$60 \cdot 10^3$	$180 \cdot 10^3$	$360 \cdot 10^3$	$600 \cdot 10^3$
mean classification-error in %:	0.81	0.67	0.59	0.54	0.55

Table 4.6: Impact of training-set size on image classification using MNIST images. For $N_{train} > 60000$ we augment the train-set by affine image transformations.

improves image classification, which was expected as this primarily facilitates learning better classifier models.

Effect of Training Images on Representation Learning: At next, we examine the influence of underlying image data that is used for representation learning. In this experiment we distinguish between two image training-sets, one for training our RF, and one for training a global classifier. Previous experiments always use the same images for both training procedures. Hence, we are encouraged to investigate the impact of different image training-sets on representation learning.

In the following experiment we perform image classification on STL-10 images [20], which are similar to CIFAR images but contain real-world object images at higher resolution (96-by-96 pixels). Further, this dataset provides three image-sets: 5000 labeled training images, 8000 labeled test images and an additional training-set of 100000 unlabeled images. This large set of unlabeled images is actually meant for evaluating unsupervised representation learning methods. Again, the goal is to classify 10 image categories where we train a classifier on the given default training-set only. The test-set is used for evaluation.

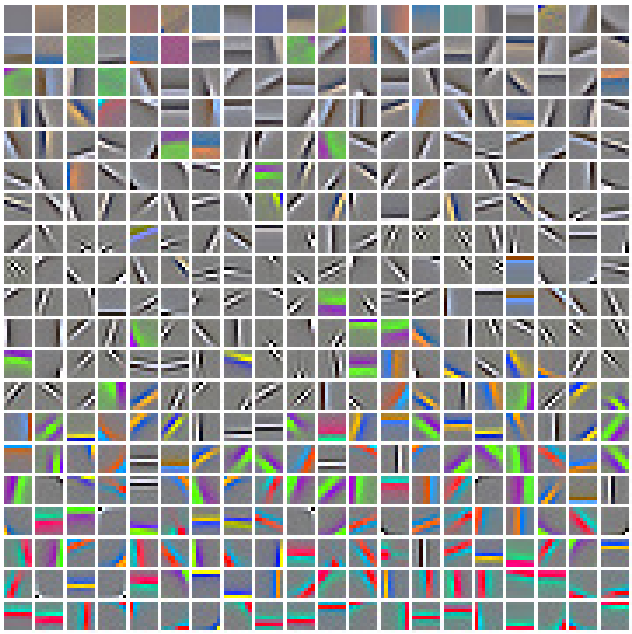


Figure 4.19: Visualization of a learned dictionary of size 400, learned from STL image patches. RGB filters sorted by variance.

RF parameter setting	
patch-size	8-by-8
patch-pool size	$M = 3 \cdot 10^6$
number of trees	$T = 10$
maximum tree depth	∞
minimum leafnode-samples	1000
post processing	
extraction stride	1
encoding activation	threshold
pooling grid-size	4-by-4
pooling operation	sum

Table 4.7: Parameter setting of RF-based representation learning for STL image classification.

Due to higher image resolution we sample image patches of size 8-by-8 and apply preprocessing using $\epsilon_{zca} = 0.01$ and $\beta = 1$. An example of a dictionary of size $K = 400$ obtained by K-means clustering is shown in Figure 4.19. To train and apply our RF we use parameters stated in Table 4.7. Note that the post-processing setting is determined empirically.

Table 4.8 shows achieved image classification results over 3 runs for different classifiers and dictionary sizes, where we employ an adaBoost forest with 512 trees and a L2-SVM. Further we compare our RF-based representations to KM-Tri representations of Coates *et*

STL-10 classification	KM-Tri		Random Forest	
Dictionary sizes	$K = 400$	$K = 800$	$K = 400$	$K = 800$
train-images, SVM	62.97	63.03	57.35	59.71
train&test-images, SVM	62.82	63.35	57.33	59.34
ul-set-images, SVM	62.80	63.03	57.65	59.45
training-images, adaBoost	69.27	**69.88**	67.17	66.80
train&test-images, adaBoost	69.28	69.26	67.11	**67.52**
ul-set-images, adaBoost	69.33	69.63	67.16	67.16

Table 4.8: Image classification results on STL images, using different datasets for representation learning. Two kinds of classifiers are trained on 5000 training-images and tested on 8000 test-images, showing the mean classification accuracy over 3 runs. For representation learning (random patch sampling) we use: the 8000 training images (train-images); the training and testing images (train&test-images); the set of 100000 unlabeled images (ul-set-images). We compare our representations with KM-Tri of Coates *et al.* [20].

al. [20]. For both, we consider using different training-sets of images for representation learning: Images of the train-set, images of the test-set and train-set, and the unlabeled image-set, all provided by default.

As it turned out, the distribution of training images that are used to learn representations in terms of dictionary learning, has no significant impact on the final discriminative power of representations. However, we observe a slight influence in the case of non-linear classification (adaBoost) for $K = 800$ where using the train&test images for RF feature learning yields a better result. This may be caused by the sampling of test-patches that are used for dictionary learning as well as for training our RF. Some test-patches are already seen and learned by the RF, which thus may affect the feature extraction on the test-set in a positive way. If we compare performance over dictionary size, we especially examine improvement for linear classification if we double the dictionary. Regarding non-linear classification, higher dimensional representations do not improve performances, still we achieve better results over linear classification.

Finally we conclude that our RF-based representations do not reach performances of KM-Tri representations which we also observe in previous experiments on CIFAR images (we refer to Section 4.3.3).

Adding RF representations to existing Experiments: We further investigate if our RF representation are able to extract additional information for existing image classification pipelines. Therefore we rerun proposed experiments of the VLFeat team [77] and append our image representations to their feature vectors. Therefore we train our RF on STL image-patches according to parameters in Table 4.7, but use an extraction stride of 8. We choose this stride because otherwise patch-based feature extraction of "higher-resolution" images would require increased processing time.

The existing experiments apply image classification on the CALTECH-101 dataset

[30] and the SCENE-67 dataset [69]. The CALTECH dataset contains 101 categories of real-world rgb-pictures that may slightly vary in size. About 40 to 800 images per category are provided, which denotes an unbalanced dataset. However, 30 random images per category are used for classifier training. The SCENE dataset contains 67 categories of indoor scenes and provides at least 100 images per class. This dataset has a total of 15620 images, again of varying size. Recall that we train our RF on STL rgb patches of size 8-by-8 where we neglect scale with respect to low-level features for high-resolution[4] images.

The VLFeat team [77] uses SIFT [58] as basic features (for further details it is referred to the work of the VLFeat team). Further, they apply the following feature encodings: (1) Bag-of-visual-words (`bovw`) use 4096 vector quantized visual words, (2) Fisher vector encodings (`fv`) [68] use 256 visual words Gaussian-Mixture-Model, and (3) Vector of Locally Aggregated Descriptors (`vlad`) [50] use 256 vector-quantized visual words. The augmentation label (`aug`) denotes that additional spatial information is encoded by appending coordinates to the representations. [77]

Optionally, we append our RF representations (`rf`) to their feature vectors. We apply K-means for dictionary learning (with patch-preprocessing parameters $\epsilon_{zca} = 0.01, \beta = 1$) and learn $K = 400$ filters. With 4-by-4 spatial pooling we therefore produce image representations of length 6400. Table 4.9 shows achieved results with or without appending our RF-based representations. We show the mean average precision (mAP and 11-point-mAP) and the mean classification accuracy for several experiments.

Datasets	**Caltech-101**			**Scene-67**		
Measures	mean Accuracy	mAP	mAP11	mean Accuracy	mAP	mAP11
`bovw-aug`	**76.26**	77.43	77.15	48.99	48.17	50.16
`bovw-aug-rf`	76.11	**77.60**	**77.39**	**50.27**	**49.85**	**51.83**
`fv-aug`	**75.96**	77.41	77.16	58.27	59.89	60.60
`fv-aug-rf`	75.37	**77.49**	**77.20**	**58.65**	**60.90**	**61.72**
`vlad-aug`	**79.75**	81.27	80.94	52.20	53.83	55.22
`vlad-aug-rf`	79.10	**81.40**	80.93	**54.32**	**54.81**	**56.04**

Table 4.9: Adding RF-based features to existing experiments of the VLFeat toolbox [77]. The affix `rf` denotes appended RF-based image representations.

As seen, by adding our RF-based representations we gain some performance improvement. The mean classification accuracies increase for the SCENE dataset, but slightly decrease for the CALTECH dataset. However, the mAP results generally improve on both test-sets, though classifying 100 categories seems to be more challenging. Recall that we use a quite large stride of 8 during feature extraction, which also equals the patch-size. Future experiments could use smaller strides (and thus encode overlapping patches) which may result in further improvement.

[4] In this context, high resolution images are approximately of size 100×100.

4.4 Discussion

In previous experiments we evaluate our RF-based feature extraction approach on different datasets and examined how representation quality is affected. At first we introduce a proof-of-concept setup that allows us to compare different image representation on image classification. Considering that we use similar image representation dimensionality for all extraction methods, we show that our RF is capable to produce well discriminative representations in the case of using MNIST images.

In general, the resulting representation quality is influenced by dictionary learning parameters and RF parameters. Through extensive experiments on the MNIST dataset we find that a high number of decision trees and a proper patch-size is critical. Further, larger dictionaries are able to describe a larger variety of features, however, this may exceed working memory.

In regard to dictionary learning we either apply K-means clustering (KM) or Sparse Coding (SC). Our experiments prove that SC facilitates learning better dictionaries over KM and thus provides more qualitative RF training-data but at higher computational cost. This equals the findings of [21, 22]. However, depending on available hardware resources SC limits the range of possible dictionary sizes. Hence, we choose to apply the simpler KM for most of our experiments.

Table 4.10 gives an overview of different image classification approaches using the MNIST dataset. As seen, compared to existing works that also employ complex deep CNN structures, our approach shows competitive performances although being a simple feed-forward single-layer system. We achieve a best overall performance on MNIST test images with an classification error of 0.54%. Such low errors occur at three experiments: (i) By using learned pipeline settings which we obtain by genetic learning, (ii) if we use a RF with 100 trees and 50 mls (we refer to the experiment in Section 4.3.2) or (iii) if we augment training data by affine image distortions and training only 32 trees. Note that for case (ii) and (iii) we learn dictionaries of size 100. Considering how we achieve the result in case (i), we conclude that it is possible to successfully learn hyper-parameters of an entire pipeline by employing meta-learning approaches.

As seen, our method is competitive to multi-layer CNNs, although we only use single-layer representations that are learned in an unsupervised fashion. Further, training data augmentation is not explicitly necessary to achieve good classification results. Also in terms of computational efficiency, our framework reveals quite low runtimes, although we do not focus on extremely efficient implementations (we refer to Section 3.2 for further details). To get an error of 0.54 % on MNIST-10 our framework requires about 10 minutes, which includes time consumption of our representation learning procedure, and of training and testing a global image classifier. The underlying system uses a Intel® Xeon® processor with 24 cores at 3.07 GHz, where during feature extraction 8 cores were busy.

Regarding image classification on more challenging datasets like CIFAR or STL, our approach does not reach the performances of Coates *et al.* [20], however, we still achieve

Image classification method on MNIST-10	Preprocessing or Distortions	Test Error (%)
RF(20 trees, $K = 36$, KM))+SVM	-	1.42
RF(20 trees, $K = 36$, KM)+adaBoost	-	0.68
RF(32 stumps, $K = 100$, KM))+adaBoost	-	1.01
RF(100 stumps, $K = 100$, SC)+adaBoost	-	0.75
RF(100 trees, $K = 100$, KM)+adaBoost	-	0.57
RF(20 trees, $K = 512$, KM)+adaBoost	-	0.57
RF(100 trees, $K = 100$, SC)+adaBoost	-	0.55
RF(32 trees, $K = 100$, KM)+adaBoost	affine distortions	**0.54**
RF(GA[a]: 57 trees, $K = 264$, KM)+adaBoost	-	**0.54**
Trainable Feature Extractor+SVM [55]	-	0.83
Trainable Feature Extractor+SVM [55]	elastic distortions	0.56
Trainable Feature Extractor+SVM [55]	affine distortions	**0.54**
Virtual SVM,deg-9 polynomial, 2pixel-jittered [26]	deskewing	0.56
CNN (LeNet-5) [56]	-	0.95
CNN (LeNet-5) [56]	distortions	0.8
Large CNN, unsup. pretraining [49]	-	0.53
Committee of 35 CNNs [19]	width-normalization, elastic distortions	**0.23**

Table 4.10: Comparison of image classification methods on MNIST data. Performance is stated as mean classification error on the test-set. Preprocessing and distortions are related to training data. Hence, some approaches augment training data by additional image distortions to increase the training-set.

[a]Genetic Algorithm result; see Algorithm 2

satisfying results despite we extract features from raw image data which is an additional advantage. Importantly, we show that our proposed method is, in contrast to the method of Coates *et al.*, less sensitive to preprocessing parameters for dictionary learning. Further, we also examine time efficiency with respect to feature extraction. Experiments reveal, that the time consumption of the RF inference is almost independent on the learned representation dimensionality. However, to obtain well discriminative image representations, post-processing of encoded data is critical. Thus, we find that the time consumption for post-processing is still an issue in our approach. To recap, compared to the classification results of Coates *et al.*, our method yields a slight performance decrease, which is acceptable as our approach is able to produce image representations from raw input data in a more efficient way.

Another experiment reveals that if we add our representations to existing image classification experiments, we observe a slight improvement. We successfully show this for experiments of the VLFeat Team [77]. Thus we conclude that our RF is capable to extract additional feature information although of being based on unsupervised learning.

5 Hybrid Feature Learning

As discussed in our previous approach in Section 4, we successfully show that Random Forests (RFs) are capable for patch-based representation learning. Since this enables efficient pixel-wise feature extraction, we propose to employ a RF (again) as a first-layer feature extractor within a Convolutional Neural Network (CNN). We denote this as a multi-layered *hybrid* architecture and aim at applying an end-to-end optimization on image classification which includes simultaneous training of a RF and of CNN layer-weights in a supervised fashion.

Nowadays, multi-layered convolutional architectures show state-of-the-art performances on various computer vision tasks. However, apart from the challenge of manually designing such networks, this also comes with high computational effort during training and testing the system. Especially the first convolutional layers often perform expensive convolutions due to relatively high input image resolutions or large filter kernels. To reduce computational cost we therefore propose to replace the first convolutional layer of a CNN by a RF, which enables a non-linear and fast first-level feature extraction. Note that feature extraction with RFs works the same way as for our previously introduced approach, where explanations are stated in Section 3. However, optimizing such a hybrid pipeline requires different considerations towards training the RF model.

During end-to-end optimization, we back-propagate the error of the network output-layer until we reach the RF-output (feature channels). In fact, "updating" a RF in terms of updating weights or parameters as it is usually done for traditional CNN structures is not possible. Thus, the goal is to compensate the RF-output error-gradient and to update the ensemble model instead. Therefore we iteratively train at least one new tree on appearing error-residuals, which is also known as Gradient Boosting [34, 35]. With increasing number of iterations, the RF grows and strengthens its capability to extract first-level features for the following network.

Section 5.1 introduces preliminaries and explain general optimization of CNN networks. The used network architectures are stated in Section 5.2. Experiments are shown in Section 5.3 followed by a discussion in Section 5.4.

5.1 Preliminaries

At first we briefly introduce some basics and explain common types of layers that are used in CNN structures. Further we explain related terms with respect to end-to-end optimization.

5.1.1 Layers of Convolutional Networks

The purpose of each network layer is to map input data to a new representation at its output. By subsequently stacking several layers for image classification, we obtain abstract representations that ideally allow to infer correct categorization from given input images. However, due to the variety of possible network structures, it is not trivial to find optimal architectures. Typical CNN layers are: Convolutional Layer (`conv`), Pooling Layer (`pool`), Rectified Linear Unit (`ReLU`) and Fully-Connected Layer (`fc`). Each of them performs different input-output mappings and is able to handle 3D data volumes which we also denote as feature channels.

Input Layer: A raw input image of size $H \times W \times q$ is denoted as input layer and is fed to the first layer of the network. In typical networks a convolutional layer extracts first-level features and produces a higher number of feature channels than given by an input image. Thus, we have to mind the resulting size of the first layer output if input images have high resolution.

Convolutional Layer: This kind of layer computes features by convolving a local region (receptive field) of an input volume with the weights of a neuron. Resulting output-values of one neuron correspond to one output value. However, if each neuron would use individual weights, computational resources may not suffice. We therefore assume weight sharing (or also called parameter sharing). This way, neurons that correspond to the same output feature map share the same weights, which is equal to using a filter kernel for convolutional feature extraction.

With a given convolution stride (step-size between spatial regions) and without certain border-handling one gets a reduced size for the output volume. The actual number of weights defined by a convolutional layer depends on the filter-size $w \times w$, the number of input-channels Q and the number of output-channels K. Then, a resulting weight matrix of such a layer spans $w \times w \times Q \times K + K$ dimensions, including a shared bias.

Pooling Layer: As seen, the size of feature volumes in-between convolutional layers may increase extensively. To circumvent this problem it is usual to apply spatial pooling which is discussed in Section 2.3. The dimensionality reduction of feature channels enables more efficient processing on following layers and additionally induces invariance to image transformations. Hence, a whole network becomes more robust while memory consumption is reduced. Again, the resulting downsampled output volume dimensions depend on the pooling-window size and its stride, but note that the number of feature channels is preserved.

Rectified Linear Unit: The ReLU simply thresholds the input data and only passes values above zero. This rectifying linear activation is mathematically formulated as $f(x) = \max(0, x)$ where x is a single input value. Hence, if the network is initialized randomly at zero-mean, the next layer sees sparse features as only half of the neurons may be activated. It has shown that this allows more efficient training for multi-layered structures [38].

Fully-Connected Layer: Classical neural networks use fully-connected layers where each input neuron is connected to each output neuron. Of course, this is only practicable for simple problems as it would easily exceed working memory concerning image data. CNNs for image processing usually use such fully connected neurons in the last few layers.

Softmax Activation: Concerning image classification, we append a softmax activation layer at the end of our CNN. That is, for K possible category outputs, we produce a distribution $p(k|\mathbf{x}) = \mathbf{y}$ of length K, following Equation 5.1. An input value $x_k \in \mathbf{x}$ corresponds to the k-th output value of the previous layer. The final class prediction is defined by $\hat{c} = \text{argmax}_k\, p(k|\mathbf{x})$.

$$y_k = \frac{\exp(x_k)}{\sum_{j=1}^{K} \exp(x_j)} \tag{5.1}$$

Normalization Layer: Various normalization methods on intermediate feature channels have been discussed [49], some trying to imitate inhibition of biological neurons. However, in practice it turned out that they do not really contribute additional improvement to modern network architectures [54]. On the other hand, if we consider using a RF in a hybrid network, normalization on the output ensures a fixed value range for the remaining CNN layers. Hence, we apply Local Response Normalization (LRN) [49] by following

$$y_{ijk} = \frac{x_{ijk}}{\sqrt{1 + \sum_{q=1}^{K} x_{ijq}^2}}, \tag{5.2}$$

where $\mathbf{x}_{ij}$ corresponds to the RF-output vector at location (i, j).

5.1.2 End-to-end Optimization

To train our proposed networks we consider applying an end-to-end optimization which uses *error back-propagation* and *gradient descent*. Therefore we aim at iteratively learning the entire pipeline, which in case of our hybrid network also includes a RF. At first, we briefly look at how to optimize weights of a classical multi-layer CNN in general.

Given L layers with individual weights $\mathbf{w}_l$, training data $\mathbf{x}$ is passed through the network up to the final output-layer (in this example a Softmax-layer), producing an output $\mathbf{y}_L$. We introduce a general layer-function $f_l(\mathbf{x}_l; \mathbf{w}_l) = \mathbf{y}_l$, where its output may be the input of a following layer $\mathbf{x}_{l+1} = \mathbf{y}_l$. A mathematical *feed-forward* formulation of L subsequent layers is shown in the following equation:

$$f_{net}(\mathbf{x}) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2)\dots; \mathbf{w}_{L-1}); \mathbf{w}_L) = \mathbf{y}_L. \tag{5.3}$$

During optimization the network output $\mathbf{y}_L$ enables to compute a *loss* and to back-propagate its gradient through all of the layers. The network-weights are then updated by a gradient descent algorithm. Note that computing the network loss requires labeled training data. Thus we denote this end-to-end optimization as supervised learning process.

Loss Function

Each optimization iteration during training requires to determine the actual network error. Therefore we apply a defined loss-function $\mathcal{L}(\mathbf{y}, \mathbf{t})$, which uses a target vector $\mathbf{t}$ (ground-truth information). That is, we either consider using the log-loss,

$$\mathcal{L}_{\log}\big(\mathbf{y}_L, \mathbf{t}\big) = -\sum_{k=1}^{K} t_k \log(y_{L,k}), \tag{5.4}$$

or using the squared-error-loss,

$$\mathcal{L}_{\text{se}}\big(\mathbf{y}_L, \mathbf{t}\big) = \frac{1}{2}\sum_{k=1}^{K}(t_k - y_{L,k})^2, \tag{5.5}$$

where $\mathbf{y}_L$ is the final softmax-output of the network. The target vector is defined to be one-hot encoded by holding a 1 at the index $k = c$ which equals the ground-truth class c of a training image. Note, $\mathbf{t}$ is of equal size as $\mathbf{y}_L$. If not otherwise stated, we define the L-th network layer as the final softmax-layer and apply the loss function on its softmax-output $\mathbf{y}_L$.

Given a batch of M training samples we then compute a batch-loss by,

$$z = \frac{1}{M}\sum_{m=1}^{M} \mathcal{L}\big(\mathbf{y}_L^{(m)}, \mathbf{t}^{(m)}\big), \qquad z \in \mathbb{R}. \tag{5.6}$$

Gradient Descent

Given a network loss, the goal is to minimize a global error-function $E(\mathbf{y}, \mathbf{t}; \Theta)$ with respect to network weights which we denote as Θ. The gradient descent algorithm enables an iterative update of Θ by following the negative gradient of the error-function, using a learning-rate α. The general gradient descent update formulation is defined by:

$$\Theta^{(i+1)} = \Theta^{(i)} - \alpha \nabla_{\Theta} E(\mathbf{y}, \mathbf{t}; \Theta) \Big|_{\Theta=\Theta^{(i)}} \tag{5.7}$$

This way we minimize the error-function as well as the network loss, and converge to the optimal setting Θ^* which is defined by,

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, E(\mathbf{y}, \mathbf{t}; \Theta). \tag{5.8}$$

Assuming a network has L layers and a loss-function $\mathcal{L}(\mathbf{y}_L, \mathbf{t})$, we introduce an error-function,

$$E(\mathbf{y}^{(j)}, \mathbf{t}^{(j)}; \mathbf{w}_l)\Big|_{j\in\{1,\dots,M\}} = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}\big(\mathbf{y}^{(m)}, \mathbf{t}^{(m)}\big) + \frac{\lambda}{2} \left\|\mathbf{w}_l\right\|^2, \tag{5.9}$$

which additionally penalizes high weight values in terms of regularization. This causes so-called *weight decay* and ensures a convex error-function, where λ is the decay-rate or shrink-rate.

We apply **stochastic gradient descent with momentum** where the stochastic behavior comes from using mini-batches, as this requires less working memory during training. Therefore we use mini-batches of M samples (random subset of training-set) which yields fluctuating loss. To further smooth the gradient update we introduce a momentum term $\mathbf{u}$ with momentum μ. The following update equations show how to iteratively compute momentum terms as well as the gradient descent update with respect to the weights of layer l, where α is the learning-rate:

$$\text{momentum term:} \qquad \mathbf{u}_l^{(i+1)} = \mu\ \mathbf{u}_l^{(i)} + \nabla_{\mathbf{w}_l} E(\mathbf{y}, \mathbf{t}; \Theta) \tag{5.10}$$

$$\mathbf{u}_l^{(i+1)} = \mu\ \mathbf{u}_l^{(i)} + \frac{1}{M} \sum_{m=1}^{M} \frac{\partial \mathcal{L}\big(\mathbf{y}^{(m)}, \mathbf{t}^{(m)}\big)}{\partial \mathbf{w}_l} + \lambda\ \mathbf{w}_l^{(i)} \tag{5.11}$$

$$\text{gradient descent:} \qquad \mathbf{w}_l^{(i+1)} = \mathbf{w}_l^{(i)} - \alpha\ \mathbf{u}_l^{(i+1)} \tag{5.12}$$

At next we discuss how to obtain a "gradient" (or more correct partial derivative) of our loss-function $\partial\mathcal{L}/\partial\mathbf{w}_l$.

Error Back-Propagation

To update the entire network we have to back-propagate the "gradient" of the network output-error with respect to each individual network weight within all layers. In regard to the nested formulation of a network-function f_{net} in Equation 5.3 and a loss z according to Equation 5.6, we apply the chain rule to formally obtain the derivatives by,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_l} = \frac{\partial z}{\partial f_{net}} \frac{\partial f_{net}}{\partial \mathbf{w}_l} = \frac{\partial \mathcal{L}(f_L)}{\partial \mathbf{y}_L} \frac{\partial f_L}{\partial \mathbf{x}_L} \frac{\partial f_{L-1}}{\partial \mathbf{x}_{L-1}} \cdots \frac{\partial f_{l+1}}{\partial \mathbf{x}_{l+1}} \frac{\partial f_l}{\partial \mathbf{w}_l} \tag{5.13}$$

$$= \frac{\partial z}{\partial \mathbf{y}_L} \frac{\partial \mathbf{y}_L}{\partial \mathbf{x}_L} \frac{\partial \mathbf{y}_{L-1}}{\partial \mathbf{x}_{L-1}} \cdots \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{x}_{l+1}} \frac{\partial \mathbf{y}_l}{\partial \mathbf{w}_l} \tag{5.14}$$

$$= \frac{\partial z}{\partial \mathbf{y}_L} \frac{\partial \mathbf{y}_L}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \cdots \frac{\partial \mathbf{x}_{l+2}}{\partial \mathbf{x}_{l+1}} \frac{\partial \mathbf{x}_{l+1}}{\partial \mathbf{w}_l}. \tag{5.15}$$

Note that although e.g. $\mathbf{x}$ may represent a data volume, we slightly abuse notations and assume each representation as vector-form.

However, instead of computing all network derivatives for each layer once again, from the loss back to a layer f_l, it is more efficient to subsequently back-propagate the output-derivative of layer $l+1$. Therefore we introduce,

$$g_l = \mathcal{L}(f_L(f_{L-1}(\dots f_{l+1}(f_l(\cdot))\dots)), \mathbf{t}) \tag{5.16}$$

which denotes the function of the remaining network from layer l to the last layer L, including the loss-function for simpler notation. Hence, regarding layer l, we back-propagate the output-derivative of g_{l+1} and compute the actual layer derivatives with respect to input $\mathbf{x}_l$ and weights $\mathbf{w}_l$:

$$\frac{\partial g_l}{\partial \mathbf{w}_l} = \frac{\partial g_{l+1}}{\partial \mathbf{x}_{l+1}} \frac{\partial f_l}{\partial \mathbf{w}_l} \tag{5.17}$$

$$\frac{\partial g_l}{\partial \mathbf{x}_l} = \frac{\partial g_{l+1}}{\partial \mathbf{x}_{l+1}} \frac{\partial f_l}{\partial \mathbf{x}_l} \tag{5.18}$$

As a result, we obtain derivatives of the actual layer l with respect to the following layers. Now we use $\partial g_l / \partial \mathbf{w}_l$ to update the layers weights by applying gradient descent, and $\partial g_l / \partial \mathbf{x}_l$ as output-derivative for the next layer $l-1$. This iterative back-propagation of derivatives actually circumvents an expensive computation of very high-dimensional Jacobian matrices.

5.2 A Hybrid Network

Before we introduce our hybrid network architecture and a baseline CNN, we address the RF training procedure with respect to the supervised end-to-end optimization.

5.2.1 Functional Gradient Descent and Gradient Boosting

Regarding our hybrid network, error back-propagation stops at the output of the first layer which is a non-derivable function f_{RF}, a Random Forest. Thus, we have to consider using an alternative optimization for this "layer", in particular the *functional gradient descent*.

As mentioned, our non-linear RF in the first layer ($f_1 = f_{RF} = \mathbf{y}_{RF}$) does not have "weights" $\mathbf{w}_{RF}$ and thus is not derivable as no derivative $\partial g_1 / \partial \mathbf{w}_{RF}$ exists. To bypass this optimization problem we consider using the functional gradient and apply Gradient Boosting [34].

Thus, to update the first layer RF we basically perform gradient descent (see Equation 5.7), however, instead of computing a derivative of f_{RF} with respect to concrete parameters, we just use the derivative of the remaining network with respect to our function f_{RF}. That is, we apply *functional gradient descent* and just use the derivative,

$$\nabla_{f_{RF}} \mathcal{L}(\mathbf{y}, \mathbf{t}) = \frac{\partial g_2}{\partial f_{RF}} \tag{5.19}$$

$$= \frac{\partial g_2}{\partial \mathbf{y}_{RF}} \tag{5.20}$$

$$= \frac{\partial g_2}{\partial \mathbf{x}_2}\Big|_{f_{RF}=f_1=\mathbf{x}_2} \tag{5.21}$$

where g_2 is a nested function of all subsequent layers (see Equation 5.16), starting at the second layer. As seen, this "functional gradient" is simply obtained by computing the derivative $\partial g_2 / \partial \mathbf{x}_2$ of the previous layer with respect to its input. This equals the derivative on g_2 with respect to our function f_{RF}, which also describes RF-output residuals. Further, we update our RF by adding a new tree h to the ensemble model f_{RF}, which is trained on these residuals. Ideally, the new tree compensates the previous ensemble error and improves the RF feature extraction performance. By using a learning-rate γ we minimize the loss, yielding the update equations,

$$f_{RF}^{(i+1)}(\mathbf{x}) \leftarrow f_{RF}^{(i)}(\mathbf{x}) + \underset{h}{\operatorname{argmin}}\, g_2\Big(f_{RF}^{(i)}(\mathbf{x}) + h(\mathbf{x})\Big) \tag{5.22}$$

$$f_{RF}^{(i+1)} \leftarrow f_{RF}^{(i)} - \gamma_i \frac{\partial g_2}{\partial f_{RF}} \tag{5.23}$$

where f_{RF} is defined by an ensemble of weighted (δ_t) decision trees:

$$f_{RF}(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T}\delta_t h_t(\mathbf{x}) = \mathbf{y}_{RF}. \tag{5.24}$$

As seen, we apply functional gradient descent by updating the RF-model using *stagewise additive modeling* [35], stated in Equation 5.22. In regard to decision tree ensembles, this is also known as Gradient Boosting [34]. Therefore, we iteratively train a new tree h on the previous ensemble prediction error, which also equals the back-propagated derivative $\partial g_2/\partial f_{RF}$. Note that this consecutive error-fitting minimizes the *internal* squared-error-loss with respect to RF Gradient Boosting. During end-to-end optimization of the network, this allows to adapt the mapping behavior of the RF simultaneously with the remaining CNN layer weights. Though, this undesirably increases ensemble size with learning progress.

During training a new tree we follow the procedure introduced in Section 3.2. We perform split-node sub-sampling [71] which yields 100 samples, and draw $d/3$ split-functions with $n_\tau = 3$ thresholds, where d denotes the input patch dimensionality. This configuration is fixed for all following experiments and is found empirically, showing a tradeoff between accuracy and efficiency with respect to decision trees with a maximum depth of 8.

5.2.2 Hybrid Network vs. Baseline Network

We first introduce a baseline CNN that is designed for classifying gray-valued MNIST-10 images [56]. Thus, our input layer is defined by 28-by-28 pixels. Further we propose to replace the first linear `conv`-layer by a non-linear RF model, which results in a RF-CNN architecture that we denote as *hybrid network*. The goal of following experiments is to investigate and compare image classification performance of both systems and to examine the influence of RF features.

Baseline CNN: Figure 5.1 visualizes a proposed baseline network which is assembled by different types of layers. The first layer convolves the image with 128 filters of size 6-by-6 and produces a 128 feature channels of size $24 \times 24 \times 128$ (no border-replacement). Then we apply normalization following Equation 5.2 which ensures fair comparison between this network and the hybrid version of it. Following this, we perform simple thresholding using a `ReLU` and apply spatial max-pooling using a cell-size of 4-by-4 and a stride of 3, which reduces the spatial dimensions to $7 \times 7 \times 128$. Further on we perform 64 convolutions per feature map which outputs a feature vector of length 64. After a `ReLU` layer, two subsequent `fc`-layers encode the previous feature-vector to a vector of length 32 and 10, respectively. Finally, a softmax-function activates the "raw" network output and produces

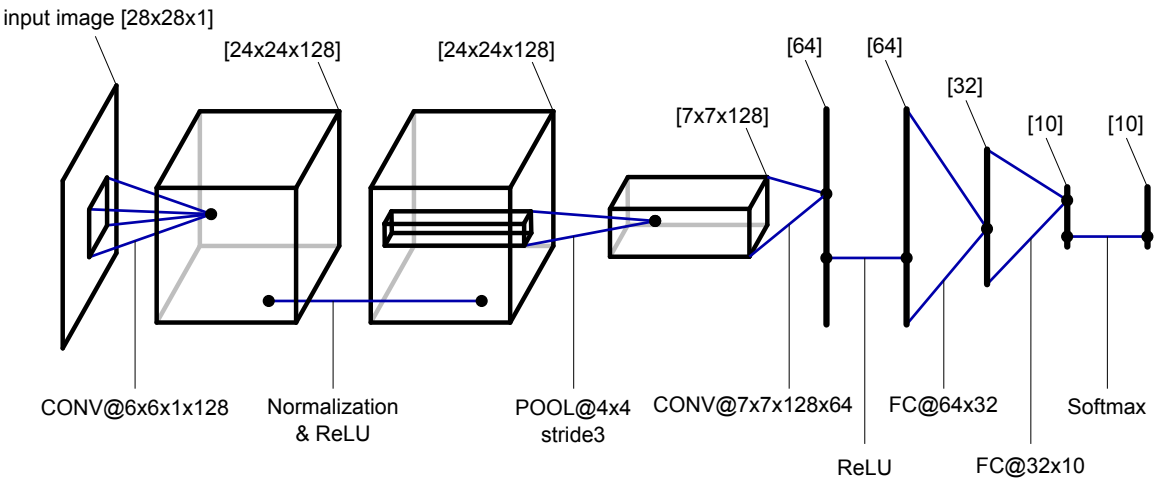


Figure 5.1: Illustration of a baseline CNN for image classification.

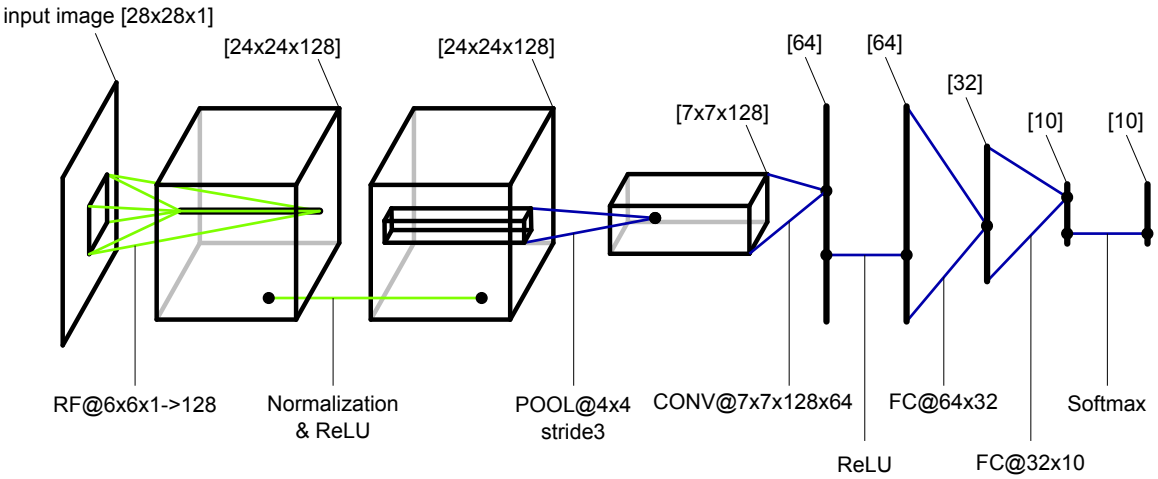


Figure 5.2: Illustration of a hybrid network for image classification. A substituted RF directly maps raw image-patches to vector representations and thus extracts first-level features.

a distribution over $C = 10$ categories, where we refer to Equation 5.1. Note that better network designs exist [19], however, the main focus of this work is not designing optimal network architectures.

Hybrid Network: Since we aim at improving our baseline CNN, we propose to replace the first convolutional layer by a RF. To preserve dimensionality on the first-layer output, RF-parameters have to be set accordingly. This includes using a patch-size of 6-by-6 and regarding 128 output feature channels, that is, trained leafnode-histograms are of length 128. As viewed in Figure 5.2, first-level features are computed by a RF that directly maps raw image-patches to feature-vectors. Because we apply a RF in a pixel-wise and patch-based manner (using a stride of 1), we obtain an output volume of size $24 \times 24 \times 128$.

During end-to-end optimization we iteratively add trees to the ensemble which affects the quality of those output feature channels for future iterations. Also note that with learning progress, the output of a RF may vary in range that may not fit to the weights of the following network layers. This requires additional normalization of the RF-output before we apply the `ReLU`, which is the reason why we also use normalization in our baseline CNN.

The detailed network setups are shown in Table 5.1, which correspond to the network illustrations in Figures 5.1 and 5.2, respectively.

	Baseline CNN	**Hybrid Network**
f_1	CONV@6x6x1x128	RF@6x6x1 $\mapsto$ 128
f_2	Normalization (Equation 5.2)	
f_3	ReLU	
f_4	POOL@4x4 stride3	
f_5	CONV@7x7x128x64	
f_6	ReLU	
f_7	FC@64x32	
f_8	FC@32x10	
f_9	Softmax	

Table 5.1: Layer setups of the baseline network and the hybrid network.

5.3 Experiments and Results

For evaluation we again perform image classification on MNIST images. We examine the performance of the baseline CNN and compare it to our hybrid network that uses RF-based feature extraction in the first-layer.

We implement our framework in MATLAB using the MatConvNet toolbox of Vedaldi and Lenc [78], which was published by the VLfeat team [77]. This toolbox provides Nvidia® CUDA® optimized implementation, including common types of CNN layers, and allows to setup and train CNN architectures in MATLAB. As stated in Section 3.2, our Random Forest is implemented in C++ by providing MEX-functionality for MATLAB. Thus, we can access the extern MEX-executables as MATLAB-functions. However, integrating our RF implementation in a MATLAB MatConvNet model exposes the following drawback: As our RF implementation efficiently handles data matrices where each image-patch is processed as data-vector, we have to transform our "feature volumes" (that describe a certain number of feature channels) of a CNN-layer to matrix form and back. Unfortunately, this inconvenient data transformations in MATLAB involves significant processing overhead that prevents fair runtime comparisons between our baseline CNN and the hybrid version. Hence, we regard our experiments as a proof

of concept where we primarily investigate the hybrid network optimization compared to the baseline network.

Due to limited memory of our GPU[1] we choose using the memory efficient MNIST image dataset, which provides a test-set of size 10000 and a training-set of size 60000. Before optimization, we additionally introduce a validation-set $\mathcal{V}$ of size 10000, which we randomly sample from the given default training-set. The remaining 50000 training images are defined as the actual training-set $\mathcal{I}_{train}$, for both pipelines. For fair comparison we train our baseline CNN and the hybrid network using the same validation-sets and training-sets.

Initialization: Regarding our baseline network, we initialize weights according to a normal distribution. Concerning the hybrid network, we also have to initialize the RF to forward data and to enable network excitation. Therefore we can either pre-train the RF, or initialize the RF randomly. To pre-train the RF we follow our first approach in this thesis and exploit the patch-based dictionary learning concept in Section 4. However, in case of random initialization we have to consider the RF output value range which is not trivial. To enable random excitation, we "train" the RF on image-patches using random label-vectors. This way the RF might learn occasional or spurious distributions that affect leafnode-statistics. Further, if leafnodes only gather a few random vectors, random outliers get more relevance which causes undesired bias. To overcome this, we assume leafnodes to hold at least 10000 samples which is quite unconventional. Since the average of an increasing number of random values converges to zero, we make a tradeoff between suppressing occasional distributions or unwanted bias, and enabling useful random excitation of the subsequent network.

For each iteration during training we sample mini-batches of $M = 500$ images. In case of updating the RF we ensure that a new decision tree "sees" enough training patches. We choose to train only one tree every 10-th iteration which we denote as RF update-period. Otherwise we could also train two or more trees per update, however, this would result in a larger final RF which is undesirable in terms of efficiency.

Note that each iteration sees a different training-data distribution due to mini-batches, which causes stochastic training.

5.3.1 Optimization with Squared-Error-Loss

Our first experiment uses the squared-error-loss for optimization, which is stated in Equation 5.5. We train the two architectures, using the parameter settings in Table 5.2. Note, the end-to-end optimization of the CNN-layers follows the update Equations 5.11 and 5.12 where we use weight decay and momentum.
As mentioned above, each training iteration uses a batch of 500 training images.

[1] We employ a Nvidia® GeForce GTX 480 with 1536 MB GDDR5 and Compute Capability 2.0

parameters CNN	
learning-rate	$\alpha = 0.1$
momentum	$\mu = 0.9$
shrink-rate	$\lambda = 0.002$
weight initialization	$\mathbf{w}_l \sim \mathcal{N}(0, \sigma = 0.1)$
training-set size	50000
validation-set size	10000
test-data size	10000
parameters RF	
initial number of trees	$T_0 = 1$
maximum tree depth	8
initial number of training images	$5 \cdot 10^4$
initial random label-vectors	$\mathbf{l}_m \sim \mathcal{N}(0, \sigma = 0.1)$
residual learning-rate	$\gamma = 500$
update period	every 10-th iteration

Table 5.2: Parameters of baseline CNN and hybrid network for MNIST classification using squared-error-loss.

Because our training-set is of size 50000, one training epoch performs 100 iterations. We optimize over 20 epochs and update the first layer RF at every 10-th iteration by adding one new tree, which finally yields an ensemble of 201 decision trees[2]. We choose this setting because of the otherwise undesirable growth of the RF during training progress. Figure 5.3 shows an example of *training error progress* for one run over 20 epochs, including both pipeline architectures. We plot the mean classification error of the validation-batch (100 samples) and the training-batch (500 samples), averaged over the previous epoch. Every 10 epochs we evaluate the test-set error on the 10000 unseen test images, which is marked for the baseline and for the hybrid network, by '+' or 'x' respectively. Additionally, we illustrate the absolute sum of occurring residuals at each RF-update and rescale them, such that the highest occurring sum defines the scale (100%). The *training loss* of this run is plotted over 20 epochs in Figure 5.4.

If we compare the training progress of our baseline CNN to the hybrid version, we see that we improve overall network performance by employing a RF as a first-level feature extractor. In this case we also produce slightly higher overfitting which can be seen by the increased gap between our validation and training error. An explanation would be the growing complexity of our RF-model. Regarding the first two epochs, we see a delayed optimization of the hybrid network compared to the baseline. Since we update our RF every 10-th iteration, RF-output residuals are growing in the early stage which is caused by the faster learning speed of the remaining network. Nonetheless, the RF prediction

[2]Note that we initialize one tree before starting optimization.

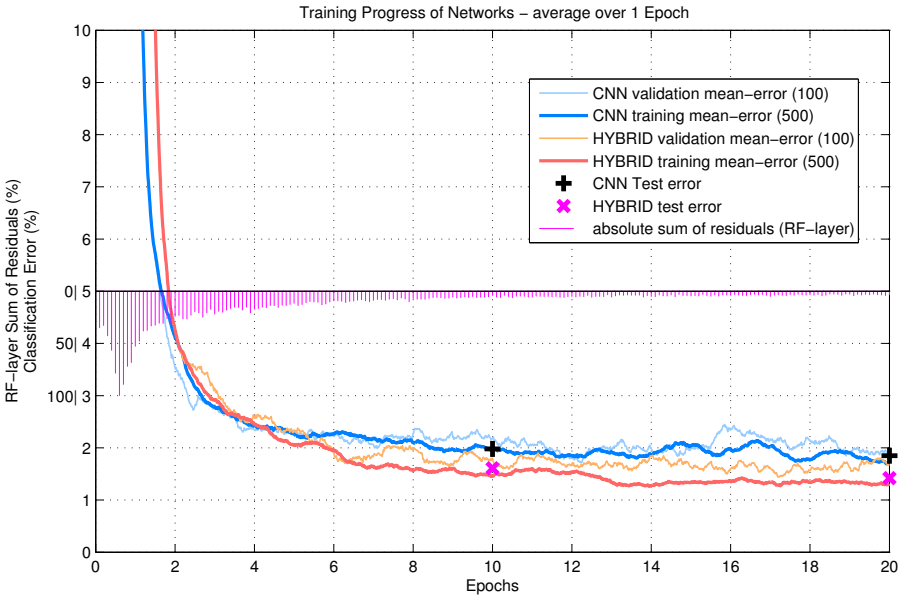


Figure 5.3: Example training error progress of both pipeline architectures, using squared-error-loss. The RF is trained with 10000 mls and depth 8. We plot the batch classification error averaged over 1 epoch (100 iterations). Blue curves and black '+' show learning progress of the baseline network. Red or orange curves and magenta markers show learning progress of the corresponding hybrid network. The magenta lines show the absolute sum of emerging residuals on the RF-output, scaled to the maximum occurring sum as 100%. (best viewed in color)

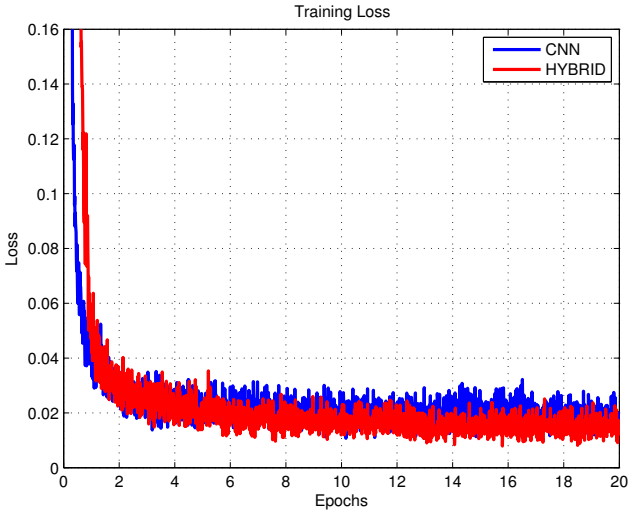


Figure 5.4: Example training loss progress of both pipeline architectures, using squared-error-loss (Equation 5.5). This plot corresponds to results in Figure 5.3

improves over time and residuals decrease. The loss of both networks settles at a certain level. The hybrid network, however, reaches a lower loss-level and thus yields improved classification performance.

In Table 5.3 we compare final results for different maximum tree depths. It states average errors over 3 runs regarding the mean batch-performances of the last epoch. Also we evaluate a pre-trained RF (starting with one tree) that exploits patch-based dictionary learning as proposed in our first approach in Section 4. As seen, we decrease the training error for all cases, however, network overfitting increases with tree depth. The lowest average test-error of 1.49% is obtained by using a pre-trained RF with depth 8, where we improve the baseline test-error by approximately 17%.

	Baseline CNN	**Hybrid Network**			
initialization		RF, depth 8 pre-trained	RF, depth 8 random	RF, depth 6 random	RF, depth 3 random
train-error	1.85%	1.21%	1.24%	1.34%	1.77%
val-error	1.92%	1.7%	1.7%	1.67%	1.9%
test-error	1.8%	**1.49%**	1.51%	1.57%	1.98%
loss	0.01968	0.01442	0.01476	0.01562	0.01931

Table 5.3: Average final mean-performances after 20 epochs, over 3 runs by using squared-error loss ('val' denotes validation). RFs are trained with depths 8, 6 and 3.

Overfitting: The hybrid networks above use RFs with 10000 minimum leafnode samples (mls), still we observe slight overfitting. Thus, the next experiment shows how our hybrid network behaves if we allow a RF with 100 mls at training depth 8. Figures 5.5 and 5.6 plot an example of obtained learning curves over 20 epochs. In this case we observe increased overfitting by our hybrid network compared to previous runs. This is reasonable as we induce higher bias which counteracts generalization during training. As a result we get an average training batch error of 0.72% and an average validation batch error of 1.92%. However, classification performance on the test-set does not decrease over the baseline CNN as we still achieve an average test-error of 1.77% for our hybrid network where the baseline network yields 1.8%.

Let us further examine and compare visualizations of occurring first-layer features after 20 epochs. In Figure 5.7 we illustrate examples of 128 feature channels for random input images, that are produced by the first `conv`-layer f_1 of the baseline network. The corresponding learned layer weights are shown in Figure 5.8 where we observe typical Edge-like or Gabor-like filters as first-level features.

In contrast, Figure 5.10 illustrates examples of RF-based feature channels from our hybrid network. In this case filters or weights do not exist and thus we only are able to investigate the RF output. As seen, although we randomly initialize the RF we learn specific features. At first sight it seems that we produce similar output as it is the case for

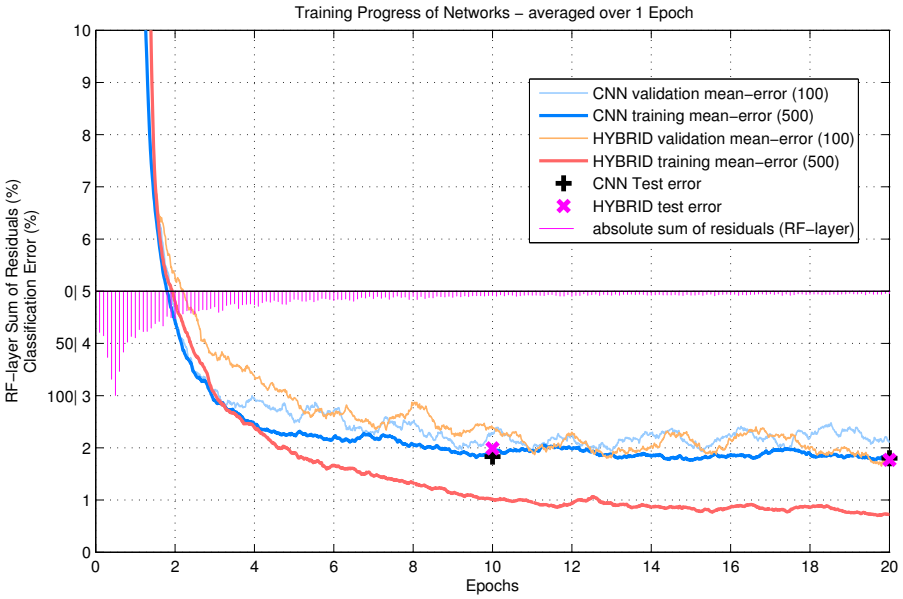


Figure 5.5: Example training error progress with overfitting, using squared-error-loss. The RF is trained with 100 mls and depth 8. We plot the batch classification error averaged over 1 epoch (100 iterations). Blue curves and black '+' show learning progress of the baseline network. Red or orange curves and magenta markers show learning progress of the corresponding hybrid network. The magenta lines show the absolute sum of emerging residuals on the RF-output, scaled to the maximum occurring sum as 100%. (best viewed in color)

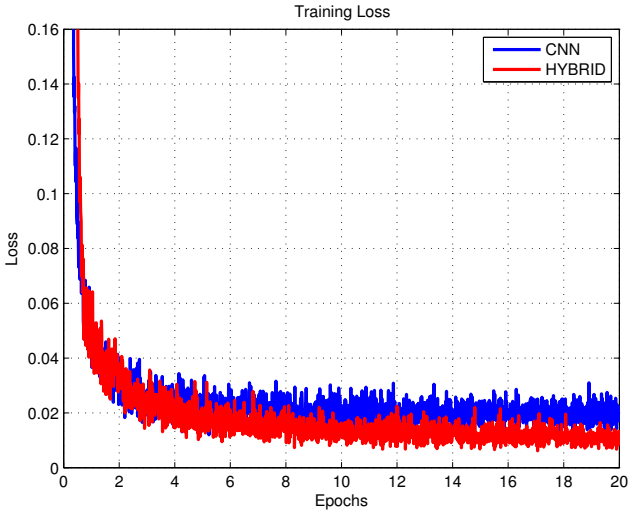


Figure 5.6: Example training loss progress with overfitting, using squared-error-loss (Equation 5.5). This plot corresponds to results in Figure 5.5

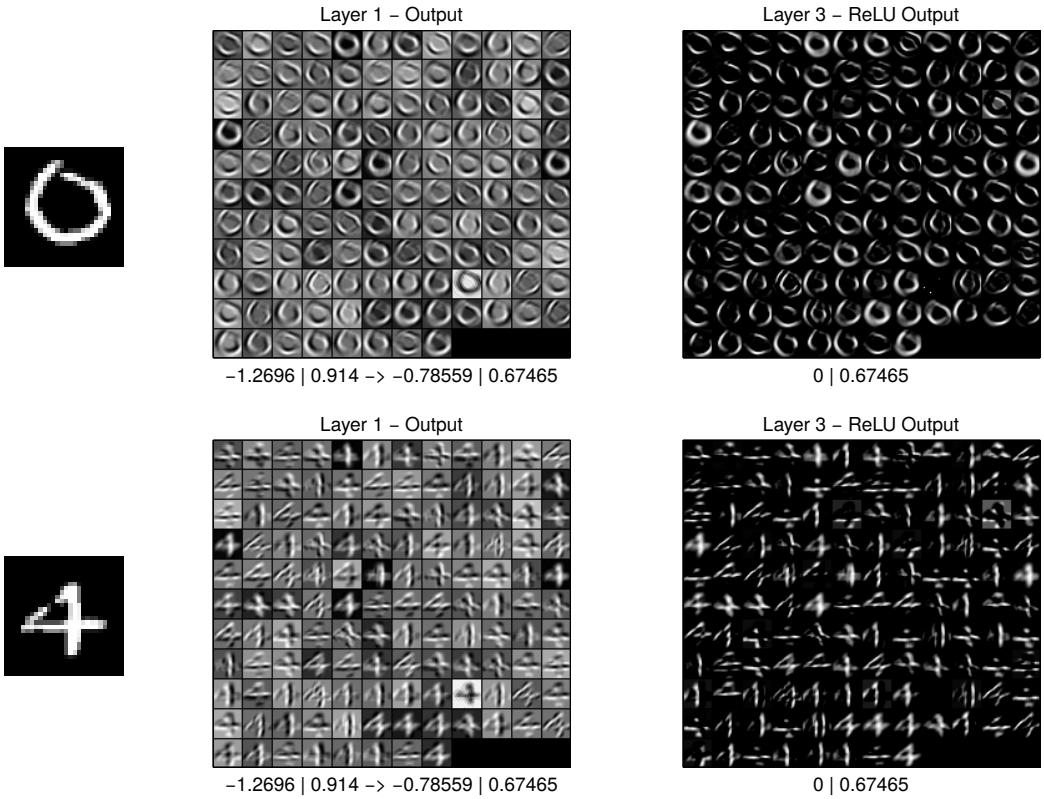


Figure 5.7: Examples of first-layer features and ReLU output of the baseline CNN for digit '0' and '4'. Bottom values denote minimum and maximum activation, the arrow indicates normalization. Left: Corresponding input images.

convolutional feature extraction with edge-like filters. A closer look reveals that some RF-based feature channels are not only activated by edges, oriented gradients or homogeneous regions, but also by combinations of such. These interesting characteristics are reasonable due to the non-linear nature of a RF-based feature extractor. However, in comparison with convolutional responses we observe that the RF outputs more noisy features. Explanations are the limited number of trees that yield slight quantization effects and thus incomplete residual compensation. Note that we obtain 201 decision trees after 20 epochs as we update our RF model at every 10-th iteration.

Layer 5 weights: Learned weights (or filters) of the convolutional layer f_5 are shown in Figure 5.9 for both architectures. If we compare both networks, we recognize that the f_5-layer weights of our hybrid CNN slightly incorporate more noise, especially regarding e.g. input channel 3. This relies on the previous RF output that also shows some noise. Also some totally random filters occur for both networks which is explainable by the following ReLU layer, preventing to properly learn some of the filters.

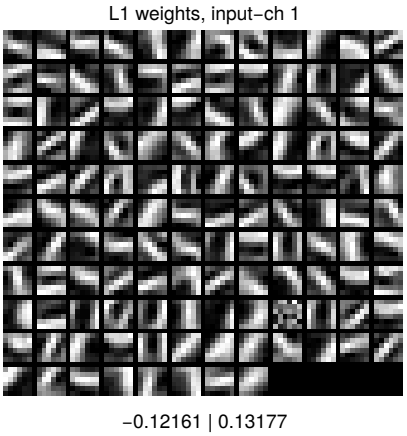


Figure 5.8: Examples of first-layer weights of the baseline CNN, learned after 20 epochs. Bottom numbers show the value range.

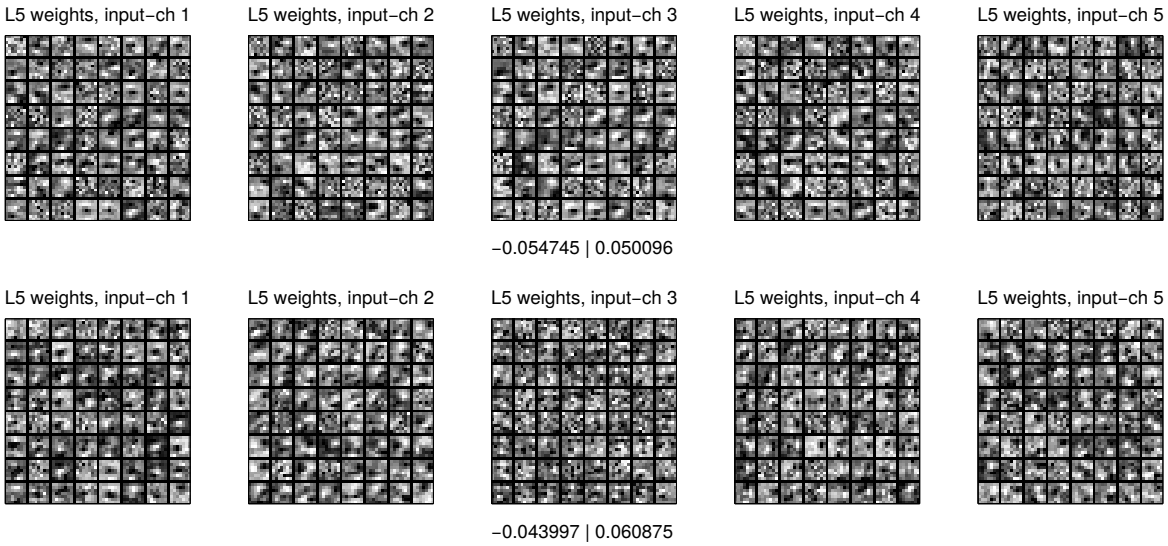


Figure 5.9: Examples of fifth-layer weights with respect to five input channels. Values at the bottom denote minimum and maximum activation of all weights. Top: Baseline network; Bottom: Hybrid network.

5.3.2 Optimization with Log-Loss

Our second experiment applies the same networks as above, but uses the Log-loss (Equation 5.4) to compute and back-propagate the gradient. However, this loss yields derivatives of different range which requires to set other learning rates and parameters, stated in Table 5.4. Comparing these parameters to the previous experiment where we use the squared-error-loss, we see major differences especially regarding learning-rates. This comes from the logarithmic nature of the log-loss which produces increased derivative magnitudes.

We again investigate both network architectures, the baseline network and the hybrid

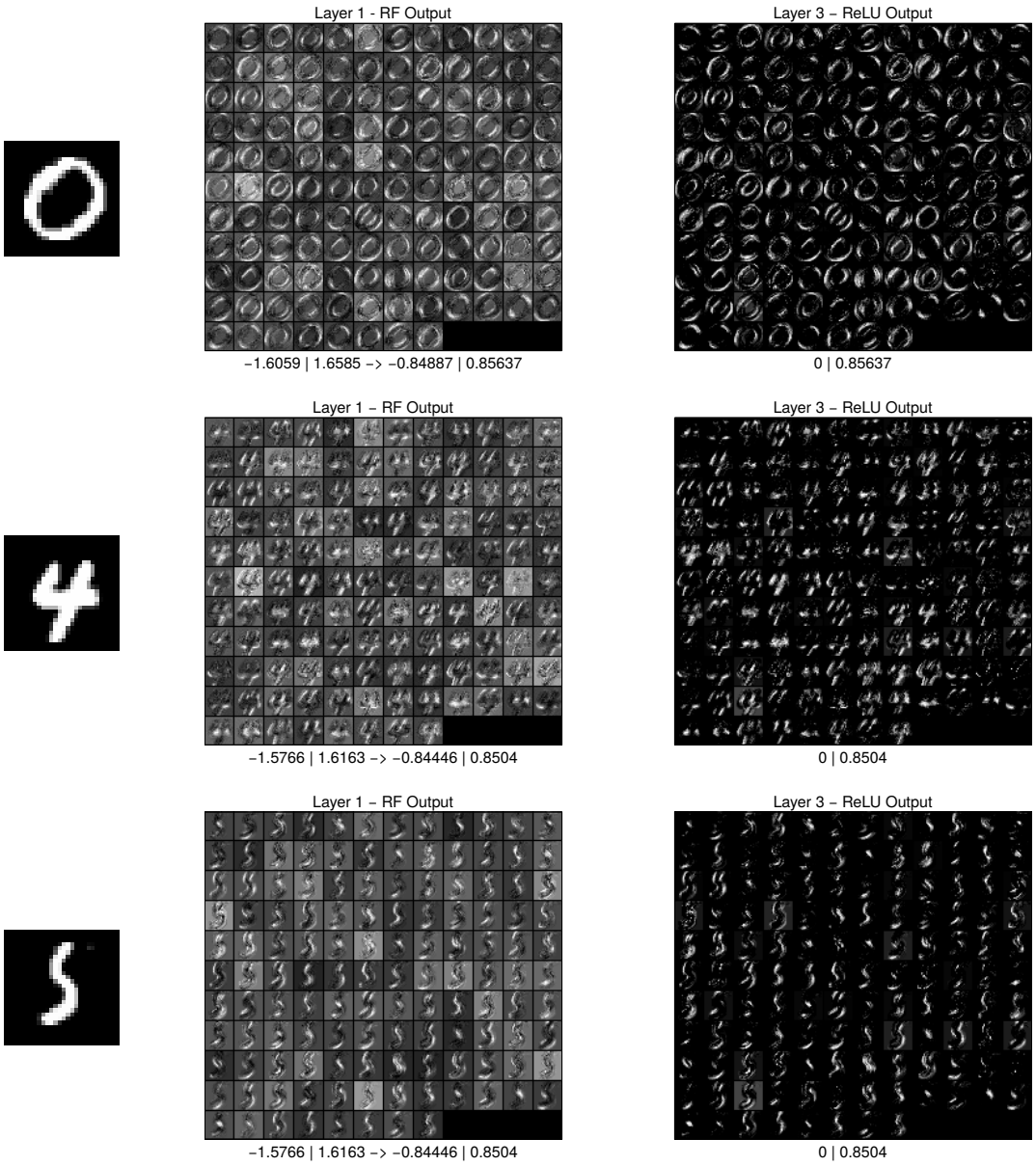


Figure 5.10: Examples of first-layer features and ReLU output of the hybrid network for digit '0','4' and '5'. Bottom values denote minimum and maximum activation, the arrow indicates normalization. Left: Corresponding input image.

network, which both are illustrated in Figures 5.1 and 5.2 respectively. An example of a learning progress over 20 epochs is shown in Figures 5.11 and 5.12, showing the learning curve and loss respectively. We use the same setup as in our previous experiment: Training-batches of 500 samples and validation-batches of 100 samples, where batch-errors are averaged over 100 iterations or 1 epoch. Final results after 20 epochs are compared in

parameters CNN	
learning-rate	$\alpha = 10^{-4}$
momentum	$\mu = 0.95$
shrink-rate	$\lambda = 0.3$
weight initialization	$\mathbf{w}_l \sim \mathcal{N}(0, \sigma = 0.1)$
training-data size	50000
validation-data size	10000
test-data size	10000
parameters RF	
initial number of trees	$T_0 = 1$
maximum tree depth	8
initial number of training samples	$5 \cdot 10^4$
initial random label-vectors	$\mathbf{v}_m \sim \mathcal{N}(0, \sigma = 0.1)$
residual learning-rate	$\gamma = 0.1$
update period	every 10-th iteration

Table 5.4: Parameters of baseline CNN and hybrid network for MNIST classification using log-loss.

	Baseline CNN	**Hybrid Network** RF, depth 8
train-error	1.98%	1.98%
validation-error	2.08%	2.26%
test-error	**1.88**%	2.14%
loss	0.08421	0.09064

Table 5.5: Average final mean-performances after 20 epochs, over 3 runs by using log-loss. RFs are initialized randomly.

Table 5.5 where we average over 3 runs.

As seen, our hybrid network does not show improved performance over the baseline network after optimizing for 20 epochs, although using the same architectures as stated in the previous experiment. Besides the possibility of improperly chosen parameters or learning-rates, we explain this by the underlying log-loss function. The logarithmic behavior of the log-loss causes a fast decrease of error gradients with training progress. This can also be seen in Figure 5.11 where we observe that magnitudes of the RF output residuals decrease really fast. Hence we conclude that the RF is not able to learn sufficient patch-based mapping, since very small residuals prevent learning effective trees and thus ongoing adaption. To sum up, the logarithmic behavior of this loss affects optimization of our *additive* RF model.

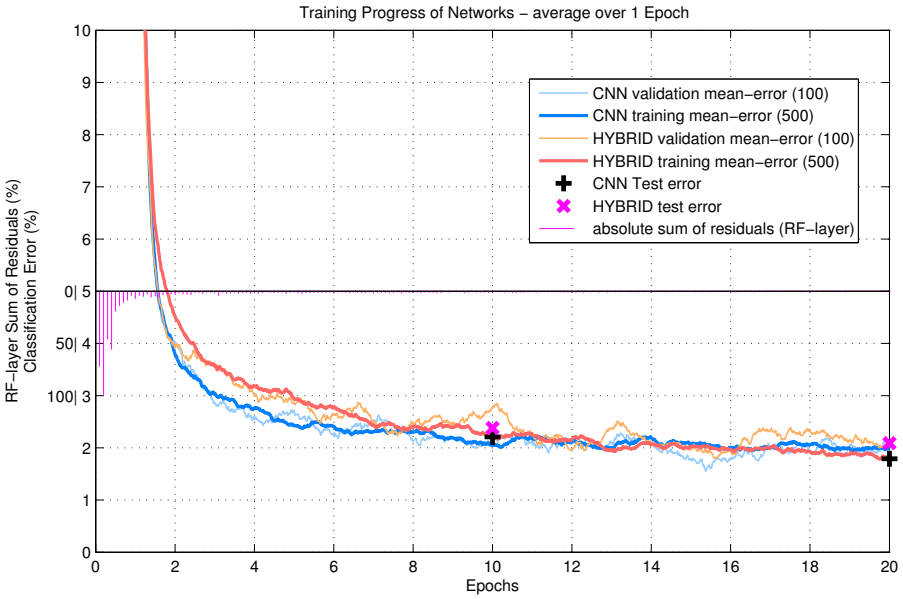


Figure 5.11: Example training error progress of both pipeline architectures, using log-loss. The RF is trained with 10000 mls and depth 8. We plot the batch classification error averaged over 1 epoch (100 iterations). Blue curves and black '+' show learning progress of the baseline network. Red or orange curves and magenta markers show learning progress of the corresponding hybrid network. The magenta lines show the absolute sum of emerging residuals on the RF-output, scaled to the maximum occurring sum as 100%. (best viewed in color)

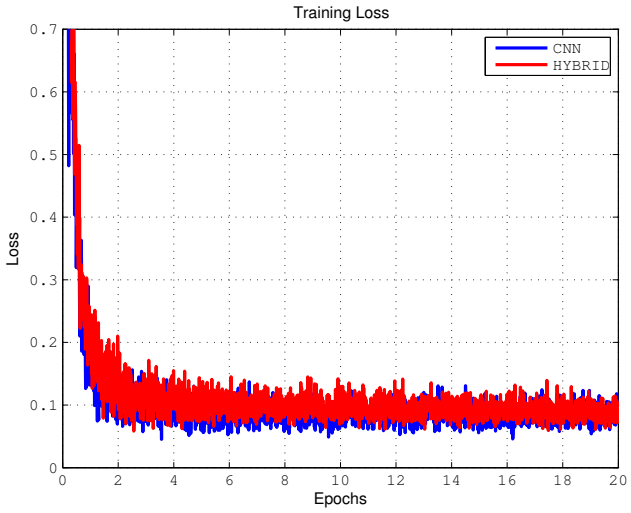


Figure 5.12: Example training loss progress of both pipeline architectures, using log-loss (Equation 5.4). This plot corresponds to results in Figure 5.11

5.4 Discussion

The idea of replacing the first convolutional layer of an existing CNN by a Random Forest is unconventional. Hence, the goal is to examine if overall image classification can be improved by this modification. Common convolutional layers of classical CNNs perform linear mapping of inputs, however, a RF is able to extract features in a non-linear way. Thus, previous experiments show that a RF as first-layer feature extractor enables improved performance under certain circumstances.

At first we have to admit that our baseline CNN does not perform very well on MNIST classification. For instance, compared to the simple LeNet-5 CNN-architecture of LeCun *et al.* [56] which shows a test-error of 0.95%, our baseline CNN only achieves approximately 1.8% although of having way more network weights. We explain this performance gap by two major differences: On the one hand LeCun *et al.* use enlarged input images of 32-by-32 pixels, whereas our approach uses the given 28-by-28 sized images. Because digits are provided within a centered region of 20-by-20 pixels by default, their border enlargement allows to capture potential distinctive features at the digit boundary, since features can appear in the whole area within a patch (or receptive field). However, we use the given default input image size for all experiments in this thesis, as this allows fair comparison between both of our feature learning approaches. On the other hand, the LeNet-5 architecture uses subsampling layers with trainable neighborhood-coefficients and sigmoid activation, whereas our networks merely perform fixed spatial pooling.

Apart from that we do not focus on state-of-the-art network performance but intend to examine the impact of a first layer RF on global image classification.

To recap our experiments, several settings are critical for certain improvement with hybrid architectures. At first, one has to care about the RF initialization, especially concerning random initialization. As we train a decision tree on random label-vectors, a leafnode averages over them. For a high number of samples the resulting histogram vector converges to zero-mean values which may cause too low network excitation; if we only allow a few samples per leafnode, we may force learning spurious distributions or unwanted bias. Thus, we make a tradeoff by using at least 10000 samples per leafnode. Experiments show that if we allow 100 samples per leafnode, we induce higher bias which counteracts generalization. This leads to overfitting but does not decrease performance compared to the baseline network. Further we observe that random initialization and RF pre-training yield quite similar network performances after training 20 epochs.

Regarding RF optimization, a larger tree depth allows to perform more splits and thus enables more informative leafnode-histograms. With learning progress, we add an increasing number of leafnode-histograms which also causes an increasing output value range. This affects the feed-forward behavior of the following (remaining) CNN since network weights are optimized with regularization. To prevent divergence and to ensure

fair comparison we additionally apply normalization on the first-layer output, for both network architectures.

In addition, learning-rates or update rules have to be chosen wisely. We empirically found that a RF update period of 10 iterations is adequate in regard to the growing tree ensemble. Note that updating with every iteration would yield a quite large RF which slows down feature extraction. Hence, tuning a hybrid network gets more challenging as many more hyper-parameters have to be found.

Furthermore, we conclude that selecting the right loss-function mainly contributes to certain success with our proposed hybrid network. As we directly update the RF according to its previous error residuals, this consecutive error-fitting equals minimizing an internal ensemble loss. If we train and add a new tree, we follow the negative gradient of the ensemble model and thus minimize the internal squared error. This optimization is conform with the global squared-error-loss examined in Section 5.3.1, which explains why using the log-loss in our last experiment fails to improve performance with a hybrid network.

We achieve a best performance improvement of approximately 17% over the baseline network, where we obtain a mean test-error of 1.5% on MNIST. Therefore we apply a first layer RF with a maximum depth of 8 and optimize the hybrid network using the squared-error-loss. Unfortunately, the limited memory of our system prevented us from answering the question, how many additional network layers we saved with our hybrid network, which could be addressed in future studies.

6 Conclusion

Random Forests (RFs) count among the most popular machine learning algorithms as they can be parallelized easily and have good generalization characteristics. Their applicability for regression as well as classification proved to be useful for various computer vision tasks, however, their popularity suffered in the last few years since Convolutional Neural Networks (CNNs) yield quite outperforming results. Still, this thesis addresses patch-based representation learning with RFs, since an efficient computation of image representations is highly beneficial, especially if hardware resources are limited. A major advantage of our method is the direct feature extraction on raw patch data. Although of extracting low-level features only, experiments on image classification showed relatively high accuracies.

In this work we introduced two different representation learning approaches, using unsupervised and supervised learning algorithms.

Our first approach was inspired by the works of Coates *et al.* [20] and introduces a novel method of learning representations whereby we exploit unsupervised dictionary learning on image patches to train a RF as a feature extractor. Therefore we examined the influence of two different dictionary learning algorithms, standardized K-means clustering and sparse coding [67]. As expected, K-means clustering turned out to be way more time efficient over sparse coding, however, dictionaries learned or representations learned with sparse coding showed to be more effective, especially regarding small dictionaries.

Both methods use standardized and whitened patch data and assign pseudo-label information in terms of code-vectors. In this context, experiments revealed that patch preprocessing during dictionary learning is critical for the convolutional feature extraction of Coates *et al.*, but has less influence on our method, if at all. Since we trained a RF on the raw patches and their corresponding code-vectors, the resulting feature extractor does not require any patch preprocessing during feature extraction. We showed that this enables a direct encoding from raw pixels and thus being more efficient over convolutional approaches. In addition, we followed Coates *et al.* and performed spatial feature pooling on resulting feature channels which proved to yield powerful image representations with

respect to image classification.

Experiments were shown for different image datasets, including gray-valued images and rgb-images. We achieved classification errors of about 0.54% on MNIST-10, a classification accuracy of 70.22% on CIFAR-10 and 67.52% on STL-10. These results are obtained by non-linear classification with an adaptive boosted forest, but also linear classification by employing a SVM showed good performance. As it turned out, our approach did not reach classification accuracies of Coates *et al.* regarding rgb-images. We explain this by their extremely well tuned pipeline where especially preprocessing parameters showed to be critical for success. In contrast, our approach proved to be almost independent to preprocessing parameters and enabled more efficient feature extraction, which makes a slight performance decrease acceptable. Additionally, our RF-based patch mapping showed to be independent on the number of feature channels with respect to time efficiency. However, to produce well discriminative image representations our method still requires post-processing, which revealed processing time dependency on the number of feature channels. Addressing this issue could be a subject of future research.

Due to the many hyper-parameters of our method, we also applied a meta-learning algorithm to learn some of them, which is based on genetic learning. This way we successfully showed that it is possible to learn entire pipeline configurations.

Furthermore we were able to improve performances of existing image classification pipelines of the VLFeat team [77]. Thus, we conclude that our single-layer representations reveal additional distinctive feature information.

As second approach we introduced a so-called hybrid network to learn low-level features. Therefor we proposed training a CNN architecture for image classification, but instead of using a convolutional layer we applied a RF as a first-level feature extractor. Again we employed the RF in a pixel-wise and patch-based manner, yielding feature channels for the subsequent network. Since our hybrid network required end-to-end optimization using error back-propagation, we updated the RF model by iteratively training a new tree on its error residuals or output derivatives. Hence, the RF feature extraction capability improved with optimization progress. As a result we observed a performance improvement of approximately 17% compared to the baseline CNN. However, this also comes with tuning additional parameters, like choosing a proper RF learning rate, update period or tree depth. Additionally, we found that an appropriate network loss-function is critical if using a RF in the first layer. Experiments indicated that a hybrid network optimization using the squared-error loss yield highest improvement over the baseline network.

In addition, RF initialization was not trivial as the network across all layers requires initial excitation. We examined two different methods: RF pre-training according to our first approach or random initialization. For the second case, we trained a decision tree on random image patches with random label-vectors whereby we allowed at least 10^4 leafnode samples for gathering leafnode statistics. It turned out that otherwise random

and spurious distributions could be learned which induce undesired bias during feature extraction. For instance, if we used a minimum of 100 leafnode samples we observed to learn higher bias which counteracts generalization and results in overfitting. Finally we conclude that replacing the first convolutional layer of a CNN by a patch-based RF can improve the network's overall performance but requires to find additional learning configurations.

All in all, our experiments showed that several RF parameters, such as the number of decision trees or the training depth, have to be chosen wisely. Also the number of output feature channels or the patch-size highly affect the final representation quality.

RF-based representations proved to have many advantages over convolutional feature extraction. A RF not only provides a very efficient way to obtain a feature extractor, but also allows to effectively compute representations by directly using raw input data. Even a forest of 32 stumps (depth = 1) yielded representations that allow a classification error below 1% on MNIST-10 test images. Considering such a highly effective feature extractor, especially mobile applications could be accelerated where in particular field-programmable gate arrays (FPGAs) could contribute to tremendously high efficiency.

Besides that, future work could investigate how to exploit our unsupervised representation learning approach to learn higher-level features. One opportunity would be to concatenate our proposed single-layer approach by applying dictionary learning on first-level feature channels.

The proposed classification pipelines exhibit many parameters that affect the overall performance; testing them all goes beyond the scope of our work. Thus, Future experiments on representation learning with RFs could examine how more sophisticated split-functions improve the representation quality. In this context it would also be interesting to learn linear leafnode models during RF training, which could result in further improvement.

Also, limited computational resources like working memory restricted us to investigate further interesting configurations, like using very large dictionaries or very large forests, which could be addressed in future research.

Bibliography

[1] Agarwal, A. and Triggs, B. (2006). Hyperfeatures - multilevel local coding for visual recognition. In *Proc. European Conference on Computer Vision*. (page 11)

[2] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185. (page 17)

[3] Aly, M. (2005). Survey on multiclass classification methods. (page 17)

[4] Antipov, G., Berrani, S.-A., Ruchaud, N., and Dugelay, J.-L. (2015). Learned vs. hand-crafted features for pedestrian gender recognition. In *Proceedings of the 23rd ACM International Conference on Multimedia*. (page 1, 9, 10)

[5] Aude, O. and Antonio, T. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175. (page 9)

[6] Bell, A. J. and Sejnowski, T. J. (1997). The 'independent components' of natural scenes are edge filters. *Vision Research*, 37(23):3327 – 3338. (page 32)

[7] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828. (page 1, 11)

[8] Bishop, C. M. (2007). *Pattern Recogntion and Machine Learning*. Springer. (page 18)

[9] Bosch, A., Zisserman, A., and Munoz, X. (2007). Image Classification using Random Forests and Ferns. In *Proc. International Conference on Computer Vision*. (page 18, 19)

[10] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Workshop on Computational Learning Theory*. (page 17, 18)

[11] Boureau, Y., Bach, F., LeCun, Y., and Ponce, J. (2010a). Learning mid-level features for recognition. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 9, 16)

[12] Boureau, Y., Le Roux, N., Bach, F., Ponce, J., and LeCun, Y. (2011). Ask the locals: Multi-way local pooling for image recognition. In *Proc. International Conference on Computer Vision*. (page)

[13] Boureau, Y., Ponce, J., and LeCun, Y. (2010b). A theoretical analysis of feature pooling in vision algorithms. In *International Conference on Machine Learing*. (page 4, 15, 16)

[14] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140. (page 19, 20, 35)

[15] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32. (page 3, 18)

[16] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification And Regression Trees*. Chapman & Hall/CRC. (page 3, 17, 18, 19, 21)

[17] Budnik, M., Gutierrez-Gomez, E., Safadi, B., and Quénot, G. (2015). Learned features versus engineered features for semantic video indexing. In *13th International Workshop on Content-Based Multimedia Indexing*. (page 1)

[18] Chatfield, K., Lempitsky, V., Vedaldi, A., and Zisserman, A. (2011). The devil is in the details: an evaluation of recent feature encoding methods. In *Proc. British Machine Vision Conference*. (page 1, 10)

[19] Ciresan, D. C., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 1, 3, 11, 63, 73)

[20] Coates, A., Lee, H., and Ng, A. Y. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Journal of Machine Learning Research*. (page vii, ix, xvii, 2, 3, 4, 10, 11, 12, 13, 16, 25, 29, 30, 31, 34, 36, 37, 38, 39, 40, 42, 46, 49, 52, 54, 58, 60, 62, 87)

[21] Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *International Conference on Machine Learing*. (page vii, ix, 1, 3, 4, 11, 12, 30, 31, 35, 36, 47, 62)

[22] Coates, A. and Ng, A. Y. (2012). *Learning Feature Representations with K-Means*. Springer. (page xvii, 1, 2, 10, 12, 24, 29, 32, 33, 34, 36, 37, 46, 49, 62)

[23] Cong, J. and Xiao, B. (2014). Minimizing computation in convolutional neural networks. In *International Conference on Artificial Neural Networks*. (page 25)

[24] Csurka, G., Bray, C., Dance, C., and Fan, L. (2004). Visual categorization with bags of keypoints. In *Proc. European Conference on Computer Vision*. (page 9, 24)

[25] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 1, 9, 12)

[26] Decoste, D. and Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190. (page 63)

[27] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*. (page 18)

[28] Dollár, P., Appel, R., Belongie, S., and Perona, P. (2014). Fast Feature Pyramids for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (page 40)

[29] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? In *Conference on Uncertainty in Artificial Intelligence*. (page 2, 11, 12)

[30] Fei-Fei, L., Fergus, R., and Perona, P. (2006). One-Shot Learning of Object Categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611. (page 37, 61)

[31] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645. (page 37, 40)

[32] Freund, Y. and Shapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139. (page 18)

[33] Frey, B. J. and Dueck, D. (2007). Clustering by passing messages between data points. *Science*, 315:2007. (page 16)

[34] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232. (page 4, 6, 65, 71, 72)

[35] Friedman, J. H., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 95(2):337–407. (page 4, 6, 18, 19, 65, 72)

[36] Gemert, J. C., Geusebroek, J.-M., Veenman, C. J., and Smeulders, A. W. (2008). Kernel codebooks for scene categorization. In *Proc. European Conference on Computer Vision*. (page 1, 10, 16)

[37] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42. (page 20)

[38] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Journal of Machine Learning Research*. (page 67)

[39] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. (page 55)

[40] Graham, B. (2014). Fractional max-pooling. In *Computing Research Repository*. (page 3)

[41] Grauman, K. and Darrell, T. (2006). Unsupervised Learning of Categories from Sets of Partially Matching Image Features. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 10)

[42] Hastie, T., Rosset, S., Zhu, J., and Zou, H. (2009). Multi-class AdaBoost. *Statistics and Its Interface*, 2(3):349–360. (page 18)

[43] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554. (page 11, 12)

[44] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507. (page 11, 12)

[45] Ho, T. K. (1995). Random Decision Forests. In *International Conference on Document Analysis and Recognition*. (page 3)

[46] Hubel, D. and Wiesel, T. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591. (page 2, 15)

[47] Hubel, D. and Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.2. (page 2, 15)

[48] Hyvrinen, A., Hurri, J., and Hoyer, P. O. (2009). *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision*. Springer. (page 13, 32)

[49] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision*. (page 63, 67)

[50] Jegou, H., Perronnin, F., Douze, M., Sánchez, J., Perez, P., and Schmid, C. (2010). Aggregating local image descriptors into compact codes. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 61)

[51] Jia, Y., Vinyals, O., and Darrell, T. (2013). Pooling-invariant image feature learning. In *Computing Research Repository*. (page 10, 16)

[52] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Master Thesis*. (page xviii, 3, 33, 37, 49)

[53] Krizhevsky, A. (2010). Convolutional deep belief networks on cifar-10. (page 2, 3, 12)

[54] Krizhevsky, A. (2012). High-performance c++/cuda implementation of convolutional neural networks. `https://code.google.com/p/cuda-convnet/`. (page 67)

[55] Lauer, F., Suen, C. Y., and Bloch, G. (2007). A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824. (page 63)

[56] LeCun, Y., Bottou, L., Bengio, Y., and Haffner (1998). Gradient-Based Learning Applied to Document Recognition. In *Institute of Electrical and Electronics Engineers*. (page xvii, 5, 11, 29, 37, 38, 63, 72, 85)

[57] LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 1, 3, 11)

[58] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 1, 9, 12, 61)

[59] Lu, W. and Tan, Y.-P. (2001). A color histogram based people tracking system. In *ISCAS (2)*. (page 9)

[60] Malinowski, M. and Fritz, M. (2013). Learning smooth pooling regions for visual recognition. In *Proc. British Machine Vision Conference*. (page xvii, 16)

[61] Memisevic, R. (2011). Gradient-based learning of higher-order image features. In *Proc. International Conference on Computer Vision*. (page xvii, 11, 12)

[62] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press. (page 10)

[63] Moosmann, F., Triggs, B., and Jurie, F. (2006). Fast discriminative visual codebooks using randomized clustering forests. In *Advances Conference on Neural Information Processing Systems*. (page xvii, 13, 14, 18, 19)

[64] Nair, V. and Hinton, G. E. (2009). 3d object recognition with deep belief nets. In *Advances Conference on Neural Information Processing Systems*. (page 1)

[65] Nowak, E., Jurie, F., and Triggs, B. (2006). Sampling strategies for bag-of-features image classification. In *Proc. European Conference on Computer Vision*. (page 11, 12, 24)

[66] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987. (page 9)

[67] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: a strategy employed by v1? *Vision Research*, 37(23):3311–3325. (page 3, 4, 11, 12, 34, 87)

[68] Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *Proc. European Conference on Computer Vision*. (page 1, 10, 61)

[69] Quattoni, A. and Torralba, A. (2009). Recognizing indoor scenes. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 37, 61)

[70] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252. (page 2)

[71] Schulter, S., Leistner, C., Roth, P. M., Van Gool, L., and Bischof, H. (2011). On-line Hough Forests. In *Proc. British Machine Vision Conference*. (page 19, 20, 27, 35, 72)

[72] Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 14)

[73] Singh, S., Gupta, A., and Efros, A. A. (2012). Unsupervised Discovery of Mid-Level Discriminative Patches. In *Proc. European Conference on Computer Vision*. (page 9, 11)

[74] Sivic, J. and Zisserman, A. (2003). Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proc. International Conference on Computer Vision*. (page 24)

[75] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. In *Computing Research Repository*. (page 11)

[76] Uetz, R. and Behnke, S. (2009). Large-scale object recognition with cuda-accelerated hierarchical neural networks. In *International Converence on Intelligent Computing and Intelligent Systems*. (page 1, 11)

[77] Vedaldi, A. and Fulkerson, B. (2008). VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`. (page xix, 6, 9, 10, 37, 60, 61, 63, 74, 88)

[78] Vedaldi, A. and Lenc, K. (2015). Matconvnet, convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*. (page 74)

[79] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learing*. (page 11, 12)

[80] Viola, P. A. and Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 9, 20)

[81] Wu, T. T. and Lange, K. (2008). Coordinate Descent Algorithms for Lasso Penalized Regression. *The Annals of Statistics*, 2(1):224–244. (page 34)

[82] Zhang, F., Du, B., and Zhang, L. (2015). Saliency-guided unsupervised feature learning for scene classification. *Institute of Electrical and Electronics Engineers*, 53(4):2175–2184. (page 12)

[83] Zhou, X., Yu, K., Zhang, T., and Huang, T. S. (2010). Image classification using super-vector coding of local image descriptors. In *Proc. European Conference on Computer Vision.* (page 10)