Deep Learning for Multi-Label Land Cover Classification

Konstantinos Karalas[a,b], Grigorios Tsagkatakis[b], Michalis Zervakis[a], and Panagiotis Tsakalides[a,c]

[a]School of Electronic & Computer Engineering, Technical University of Crete, Chania, Greece;
[b]Institute of Computer Science, Foundation for Research and Technology, Heraklion, Greece;
[c]Department of Computer Science, University of Crete, Heraklion, Greece

ABSTRACT

Whereas single class classification has been a highly active topic in optical remote sensing, much less effort has been given to the multi-label classification framework, where pixels are associated with more than one labels, an approach closer to the reality than single-label classification. Given the complexity of this problem, identifying representative features extracted from raw images is of paramount importance. In this work, we investigate feature learning as a feature extraction process in order to identify the underlying explanatory patterns hidden in low-level satellite data for the purpose of multi-label classification. Sparse autoencoders composed of a single hidden layer, as well as stacked in a greedy layer-wise fashion formulate the core concept of our approach. The results suggest that learning such sparse and abstract representations of the features can aid in both remote sensing and multi-label problems. The results presented in the paper correspond to a novel real dataset of annotated spectral imagery naturally leading to the multi-label formulation.

Keywords: Remote sensing, feature learning, representation learning, autoencoders, sparse autoencoders, deep learning, multi-label classification, modis, corine.

1. INTRODUCTION

The performance of machine learning algorithms is heavily dependent on the choice of data representation (features) on which they are applied[1], an observation that is particularly evident in computer vision tasks, where carefully designed hand-crafted features, such as Scale Invariant Feature Transform (SIFT) or Histogram of Oriented Gradients (HOG), have shown great effectiveness in a variety of tasks. The main drawback of these descriptors is that significant human intervention is required during their design. Furthermore, such features are highly domain-specific and have limited generalization ability. This motivates the need for efficient feature representations extracted automatically from data through *representation learning*[1], a set of techniques which intends to learn useful (*i.e.*, discriminative, robust, smooth) representations of the raw data for the purpose of higher level tasks (*e.g.*, classification, recognition) and minimize the dependency of learning algorithms on feature engineering.

Learning such features is especially difficult in problems where the underlying data are subject to many factors of variation[2]. For example, in a speech recognition task, the factors might be the gender of the speaker and the background noise. In remote sensing, there are also analogous factors including ground environmental conditions as well as cloud contamination, forming a domain full of challenges. In this work, we aim to find "good representations" for satellite data under a real-world scenario. In detail, we are interested in land cover classification, a highly significant topic for the understanding of climate and biodiversity dynamics, through a multi-label learning approach. Land cover classification is typically treated as a single-label problem, where a remote sensing pixel is associated with a particular label or class; however, pixels of the acquired images usually encode a mix of materials, due both to instrumentation and physical interactions of light. The situation where a specific example is associated with multiple labels is a well-known machine learning paradigm, the multi-label classification problem[3,4], with numerous applications in text, image, audio and bioinformatics classification.

Further author information: (Send correspondence to K.K.)
E-mails: K.K.: kkaralas@isc.tuc.gr, G.T.: greg@ics.forth.gr, M.Z.: michalis@display.tuc.gr, P.T.: tsakalid@ics.forth.gr

The key novelty of this work is that we combine the real-life problem of multispectral image annotation through multi-label learning, with innovated ideas from the representation learning theory. More specifically, we focus on a particularly successful unsupervised representation learning approach, by considering the framework of *sparse autoencoders*[5,6], a type of artificial neural network which employs nonlinear codes and imposes sparsity constraints for representing the original data. The proposed scheme utilizes a series of stacked sparse autoencoders in order to train a deep model in the context of multi-label classification. In this context, image annotation is associated with land cover, obtained through real ground-truth data collected by the European Environment Agency. The end-to-end design of the proposed scheme is composed of a three-stage pipeline consisting of:

- preprocessing and normalization of the features.
- feature-mapping using sparse autoencoders.
- multi-label classification through the learned feature-mapping.

Through our analysis, we have experimented with several options for each module, trying to evaluate the impact of them to the final performance estimation.

The rest of the paper is organized as follows. Section 2 gives a brief review of related approaches from the literature. In Section 3, we present the basic theory of autoencoders, followed by their sparse variant used in a single- and a multi-layer way. In Section 4, we describe the multi-label classification algorithms that are incorporated at the top of our system. Section 5 provides an overview of the dataset used, the performance evaluation measures and the experimental setup. Section 6 demonstrates and discusses the experimental results, whereas in Section 7 we conclude the paper.

2. RELATED WORK

In general, representation learning encompasses a variety methods, most of them based on neural networks that combine linear and nonlinear transformations of the data. This way, autoencoders (or autoassociators) were adopted with impressive success as feature learning architectures, although they were initially studied in the late 80's as a technique for dimensionality reduction by considering a hidden layer with fewer units compared to the input (forming a bottleneck). More recently, extending their initial use, overcomplete basis vectors have been employed to obtain more expressive representations, where the number of features exceeds the number of raw inputs. In this setting, a form of regularization during autoencoder learning is needed in order to avoid trivial solutions where the autoencoder could reconstruct the input perfectly, without needing to extract any meaningful features. Several autoencoder variants have been developed in order to introduce regularization in the latent space, including the denoising[7], the contractive[8], the saturating[9], and the sparse[5,6] autoencoder.

Apart from modifying the regularization penalty term, effort has also being given on the investigation of the impact of other choices on system performance, especially in terms of the network architecture. For instance, recursive networks[10] apply the same set of weights recursively over a structure (directed acyclic graphs), recurrent networks[11] where connections between units form a directed cycle, convolutional networks with whitening transformation and pooling operations for visual tasks[12], and neural networks with rectified hidden units[13].

While it has been shown that one hidden layer can approximate a function to a very high level of precision, this approach becomes impractical due to the large number of the required computational units[14]. Inspired by the human cognitive system, researchers have tried to incorporate depth into learning algorithms, which would allow to achieve function representation more compactly[15], and obtain increasingly more abstract representations. Although theoretical results have been encouraging, in practice, it has been impossible to train sufficiently deep architectures, since gradient-based optimization methods starting from random initial weights tended to get fixated near poor local optima[16].

Deep learning was revolutionized in the past decade, when the strategy of greedy layer-wise unsupervised "pretraining" followed by supervised fine-tuning was introduced[5,17]. This technique was first applied using Restricted Boltzmann Machines (RBMs) for a digit recognition task, but has proved to be an efficient approach by incorporating autoencoders in various contexts too. Nevertheless, one should keep in mind that deep architectures

do not guarantee a superiority over shallow architectures for every type of problem[18], although the behavior in specific settings is under extensive investigation. We should note that the ideas underlying deep learning have been motivated by the way the human brain seems not only to be organized, but also to process received stimuli, which is accomplished through a chain of multiple transformation stages[14]. For example, it has been experimentally shown that for the object recognition tasks, representations produced by deep architectures can resemble those features observed in the first two stages of the visual cortex, *i.e.* edges and shapes detected by the receptive fields of neurons in V1 and V2 areas.

3. FEATURE LEARNING FRAMEWORK

In this section, we present the formulation of the autoencoders scheme, one of the fundamental paradigms for unsupervised feature learning. More specifically, we investigate sparse autoencoders and how they can be applied in the concept of deep learning.

3.1 Single-layer sparse autoencoders

A classical autoencoder is a deterministic feed-forward artificial neural network comprised of an input and an output layer of the same size with a hidden layer in between, as illustrated in Figure 1a. Typically, the model is trained with backpropagation[19] in a fully unsupervised manner, aiming to learn an approximation $\hat{\boldsymbol{x}}$ of the input which would be ideally more useful compared to the raw input.

The feature mapping that transforms an input pattern $\boldsymbol{x} \in \mathbb{R}^d$ into a hidden representation $\boldsymbol{h}$ (called code) of k neurons (units), is defined by the *encoder* function:

$$f(\boldsymbol{x}) = \boldsymbol{h} = \alpha_f(\boldsymbol{W_1 x} + \boldsymbol{b_1}), \tag{1}$$

where $\alpha_f : \mathbb{R} \mapsto \mathbb{R}$ is the *activation function* applied component-wise to the input vector. The activation function is usually chosen to be nonlinear; examples include the logistic sigmoid and the hyperbolic tangent. Recently, there is a growing interest in Rectified Linear Units (ReLU), which seem to work better in supervised recognition tasks. The activation function is parametrized by a weight matrix $\boldsymbol{W_1} \in \mathbb{R}^{k \times d}$ with weights learned on the connections from the input to the hidden layer and a bias vector $\boldsymbol{b_1} \in \mathbb{R}^k$. The network output is then computed by mapping the resulting hidden representation $\boldsymbol{h}$ back into a reconstructed vector $\hat{\boldsymbol{x}} \in \mathbb{R}^d$ using a separate *decoder* function of the form:

$$g(f(\boldsymbol{x})) = \hat{\boldsymbol{x}} = \alpha_g(\boldsymbol{W_2 h} + \boldsymbol{b_2}), \tag{2}$$

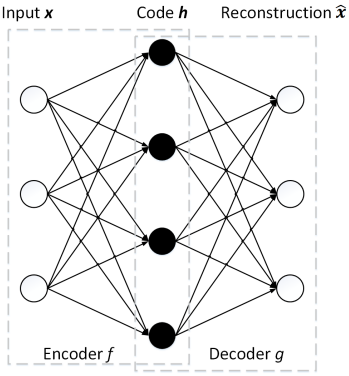
where α_g is the activation function, $\boldsymbol{W_2} \in \mathbb{R}^{d \times k}$ is the decoding matrix and $\boldsymbol{b_2} \in \mathbb{R}^d$ a vector of bias parameters which are learned from the hidden to the output layer.

The estimation of the parameters set $\theta = \{\boldsymbol{W_1}, \boldsymbol{b_1}, \boldsymbol{W_2}, \boldsymbol{b_2}\}$ of an autoencoder, is achieved through the minimization of the reconstruction error between the input and the output according to a specific loss function. Given the training set, a typical loss function seeks to minimize the normalized least squares error, defining the following optimization objective:

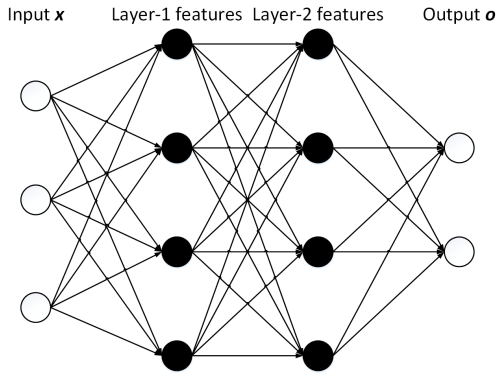
$$J_{\mathrm{AE}}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| \boldsymbol{x_i} - \hat{\boldsymbol{x}}_{\boldsymbol{i}} \right\|^2 \right), \tag{3}$$

where m is the number of training examples and $\|\cdot\|$ is the Euclidean distance. The reconstruction $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ is implicitly dependent on the parameter set θ. More advanced loss functions can also be involved[7]. A weight decay term commonly introduced to the cost function in order to prevent overfitting, has been found to influence marginal our data.

Sparse autoencoders are a special case of the typical autoencoders, where the code is constrained to be sparse, *i.e.* only a small fraction of units are active during training. Signal and model sparsity have had a profound impact on signal processing and machine learning due to their numerous advantages, such as robustness, model complexity, generative and discriminative capabilities among others[20,21]. Furthermore, evidence from neuroscience suggest that sparse networks are closer to biological neurons' responses, since the percentage of neurons being active at the same time is estimated between 1 and 4% of the total[22,23].



(a) Architecture of an autoencoder with an overcomplete hidden layer. The encoder takes the input $\boldsymbol{x}$ and computes a prediction of the best value of the latent code $\boldsymbol{h}$. The decoder is symmetric to the encoder and computes a reconstruction $\hat{\boldsymbol{x}}$ from $\boldsymbol{h}$.



(b) A 4 layer autoencoder network [3-4-4-2], where the circles denote the feature units. The black color is used to denote the hidden, whereas the white the visible units. The two middle layers constitute an encoder.

Figure 1: The autoencoder concept. The bias units are not considered for simplicity.

In order to induce the sparsity constraint, we define a sparsity constant ρ and enforce the average latent unit activation to be close to the value of ρ. This is achieved by penalizing it with the Kullback-Leibler (KL) divergence, a function employed to measure the difference between Bernoulli distributions, namely the expected activation over the training set of hidden unit u ($\hat{\rho}_u$) and its target value (ρ) in our case:

$$\mathrm{KL}(\rho||\hat{\rho}_u) = \rho \log \frac{\rho}{\hat{\rho}_u} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_u}, \quad \hat{\rho}_u = \frac{1}{m} \sum_{i=1}^{m} \left[f_u \left(\boldsymbol{x_i} \right) \right], u = 1, \ldots, k, \tag{4}$$

where $f_u(\boldsymbol{x_i})$ denotes the activation of hidden unit u. The KL distance reaches its minimum of 0 when $\hat{\rho}_u = \rho$, and extends up to infinity as $\hat{\rho}_u$ increases, enforcing the $\hat{\rho}_u$ not to significantly deviate from the desired sparsity value ρ. All in all, the smaller the value of ρ, the sparser the representation would be. The regularized cost function of a sparse autoencoder constitutes of the reconstruction loss of a classical autoencoder with an additional regularization through a *sparsity promoting term*[24] given by:

$$J_{\mathrm{spAE}}(\theta) = J_{\mathrm{AE}}(\theta) + \beta \sum_{j=1}^{k} KL(\rho||\hat{\rho}_u), \tag{5}$$

where the hyperparameter β determines the importance of the sparsity regularizer. Note that there have been also developed and other techniques to encourage sparsity in the representation[25].

A particular set of weights is updated by calculating the partial derivatives of J_{spAE} and applying the backpropagation algorithm[19]. This way, the training typically converges to a minimum, hopefully a global one, after a small number of iterations. The minimization of the model parameters θ can be achieved by conventional optimization algorithms (*e.g.*, gradient descent), as well as with more sophisticated procedures, such as conjugate gradient and Broyden-Fletcher-Goldfarb-Shanno (BFGS) methods to speed up convergence.

3.2 Deep learning with stacked sparse autoencoders

Deep learning is a special case of representation learning which admits the property that multiple levels of representations are learned hierarchically, leading to more generic and beneficial features. Ultimately, the activity of the first layer neurons corresponds to the low-level features of the input, while higher-level abstract concepts are encoded in the subsequent hidden layers. More specifically, we provide the deep architecture with surface reflectance input data, which are the raw data collected from a remote sensing observation system, and try through a hierarchical approach to learn an "advanced" version of them, which would ideally match the capabilities of

high quality hand-crafted features, such as Normalized Difference Vegetation or Enhanced Vegetation Indices (NDVI/EVI). In this way, we aim at bypassing the requirements of empirical design of these features by an expert and automatically learn representations which can substitute and enhance them. In parallel, due to the unsupervised nature of the processing, the proposed approach is more universal and could also work with other types of targets which are not chlorophyll or water sensitive, such as structures in urban areas, where analogous ratios have not been defined.

Architectures with two or more hidden layers can be created by stacking single-layer autoencoders on top of each other as depicted in Figure 1b. Formally, one starts by training a sparse autoencoder with the raw data as input. Then the decoder layer is discarded so that the activations of the hidden units (layer-1 features) become the visible input for training the second autoencoder layer (feed-forward), which in turn produces another representation (layer-2 features). This greedy layer-by-layer process keeps the previous layers fixed and ignores interactions with subsequent layers, thus dramatically reducing the search over the parameter space. While this process can be repeated multiple times, rarely more than three hidden layers are involved. We can formalize a stacked autoencoder according to:

$$\boldsymbol{h_L} = f_L\left(\cdots f_2\left(f_1\left(\boldsymbol{x}\right)\right)\right), \tag{6}$$

where $\boldsymbol{h_L}$ denotes the representation learned by the top layer L. The output of the entire architecture can be used to fed a stand-alone classifier, offering an improved representation of the data compared to the raw input.

The challenge in deep learning is that the gradient information is difficult to pass efficiently through a series of randomly initialized layers, since a good starting point is hard to identify. Unsupervised pretraining[17] is a recently developed yet very influential protocol that helps to alleviate this optimization problem by introducing prior knowledge for initializing the weights of each layer, allowing gradients to "flow well". Autoencoders, being a fundamental example of unsupervised learning, have attracted a lot of attention as a method for pretraining deep neural networks. Formally, we use the sparse autoencoder as the building block to train one layer at a time, in a bottom up fashion, for a fixed number of updates (epochs). Up until this point, the procedure is completely unsupervised. Supervised refinements are subsequently introduced in the top layer of the deep architecture in order to fine-tune the gradient-based optimization algorithm with respect to a supervised criterion, a process termed fine-tuning phase[15]. As a last optional training stage, it is possible to further optimize the parameters with a global fine-tuning, which uses backpropagation through the whole network architecture at once, however starting from a very good initial model.

3.3 Data preprocessing

A critical aspect of sparse autoencoder models is the need for data normalization. To that end, several normalization steps are usually performed in order to adapt the raw data into appropriate inputs for neural networks. Experimental results have shown that when the input variables are close to zero, neural network training is typically more efficient since convergence is faster and the likelihood of getting stuck in local optima is reduced. Formally, let the multi-label training set $X = \{\boldsymbol{x_i}, \boldsymbol{y_i} | i = 1, \ldots, m\}$, where $\boldsymbol{y_i}$ is the actual labelset of the i-th instance, $\Lambda = \{\boldsymbol{\lambda_t} | t = 1, \ldots, n\}$ is the set of all labels, and the j-th feature of $\boldsymbol{x_i}$ is $x_i^j, j = 1, \ldots, d$. We consider normalization of each feature vector j to $[0, 1]$ by subtracting the minimum value of each element and dividing it by its range (the difference between the maximum and the minimum value):

$$x_i^j = \frac{x_i^j - \min^j}{\max^j - \min^j}, \tag{7}$$

where the minimum values and ranges are stored for later use.

4. MULTI-LABEL CLASSIFICATION

The purpose of the representation learning system is to be incorporated into a data classification framework. Typical classification approaches are focused on the single class classification problem, where each training and testing example is associated with a single-label (or else belongs to a single class). In many real-life scenarios however, this is not the case. The illustrative example we consider, is labeling multispectral satellite data with ground-gathered measurements in an effort to provide up-to-date land cover usage. Due to the difference in

scale, each multispectral pixel may be associated with multiple labels, naturally leading to the case of multi-label annotation. In this work, we consider state-of-the-art multi-label classifiers that operate not on the original raw data, but in features extracted through the stacked autoencoder network, hoping to reach or even overcome the classification performance achieved by hand-crafted features.

A typical strategy to deal with a multi-label classification problem, is to decompose the original multi-label problem into a set of binary classification problems and acquire predictions through conventional single-label classification algorithms, a method known as *problem transformation*[3]. The most representative examples of problem transformation methods is the Binary Relevance (BR) and the Label Powerset (LP) techniques. According to the former method, a single-label classifier is trained independently for each label leading to a set of n classifiers, whose union forms the final prediction, whereas with the latter, each distinct subset of labels that exists in the training set, is regarded as a different class of a new single class problem.

Recently, problem transformation techniques have involved in *ensemble methods*, such as RAndom k-labELsets (RAkEL)[26] and Ensemble of Classifier Chains (ECC)[27], in order to achieve even higher classification performance. RAkEL randomly breaks the initial set of labels into a number of small subsets and then for each labelset trains a multi-label classifier using the LP technique. From the other side, ECC extends the Classifier Chains (CC) model[27] that transforms a multi-label learning problem into a chain of n BR classifiers. Although CC schema manages to take into account label dependencies, it runs the risk of low classification accuracy, since it is strongly dependent on the label order. ECC reduces this risk and achieves predictive completeness by building an ensemble of chains, each with a random label order.

Both of these techniques can be used with any off-the-shelf binary classifier. In this work, we incorporate Support Vector Machines (SVM)[28] as the base classifier, which is considered as one of the most efficient classifiers for remote sensing data. Given that each training example $\boldsymbol{x_i}$ is associated with a binary label $y_i \in \{0, 1\}$, the SVM classifier tries to find a linear separating hyperplane with the maximal margin in this higher dimensional space. Formally, when the kernel function is linear, the SVM seeks a solution to the following constraint optimization problem:

$$\begin{aligned} \underset{\boldsymbol{w}, w_0, \boldsymbol{\xi}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{w}^\intercal \boldsymbol{w} + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}\boldsymbol{x_i} - w_0) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i, \end{aligned} \tag{8}$$

where the slack variables ξ_i measure the degree of misclassification of the data, and the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin.

5. EXPERIMENTAL SETUP

5.1 Dataset

In this work, we consider the introduction of a multi-label learning scheme, adapted to a remote sensing application with real complex data. Formally, we combine real satellite data from Moderate Resolution Imaging Spectroradiometer (MODIS) instrument and high spatial resolution ground data from the CORINE Land Cover (CLC) project developed by the European Environment Agency (EEA). More precisely, the features are obtained from the MODIS sensor aboard Terra satellite*, where we consider the 7 surface reflectance bands from the MOD09A1 product, acquired at 500m^2 spatial resolution and having an 8-day revisit frequency. We seek for land cover classification, thus we collect all available data from the growing season (May to October) leading in 161 spectral bands in total. We underline that we are particularly interested in this feature set, since these are the data which are provided directly from a satellite imaging system and thus can be obtained and be accessible in short time, without the need of extra processing. Moreover, in this paper we focus in deep learning, thus we have to provide our system with primitive data in order to be able to discover the more explanatory factors hidden in that data, since by incorporating hand-crafted features the hierarchical structure of the data needed for deep learning is lost due to their inherent complex makeup, and no extra valuable information can be revealed.

*https://lpdaac.usgs.gov/data_access/data_pool

Regarding the associated ground-truth map for these inputs, we take advantage of the CORINE map[†] by the EEA, where we select data from the year 2000 annotated with 20 labels, whereas the region of interest corresponds to the h19v04 tile of MODIS. Note that CORINE has a higher resolution of 100m^2 than MOD09A1 product, which naturally leads to the multi-label case, since each spectral pixel is associated with multiple CLC codes. In accordance with multispectral and hyperspectral image single-label classification, we aim at classification with limited training examples.

5.2 Performance evaluation

The performance evaluation of multi-label classifiers is more complicated than conventional single-label learning, since an example may be partially correct. As a consequence, several metrics have been proposed for classification and ranking[4]. In this work, we quantify the performance in terms of the following six state-of-the-art metrics.

Let q be the number of testing examples in the multi-label dataset and $\boldsymbol{z_\omega}$ is the predicted set of labels. Then the example-based hamming loss is calculated by:

$$\text{Hamming Loss} = \frac{1}{q}\sum_{\omega=1}^{q}\frac{|\boldsymbol{y_\omega}\Delta\boldsymbol{z_\omega}|}{n}, \tag{9}$$

where Δ stands for the symmetric difference (corresponds to the XOR operator in Boolean logic) between the two sets. Conceptually speaking, hamming loss measures how many times a relevant label to an example is not predicted, or an irrelevant is incorrectly predicted, reaching its best performance at score 0 and worst score at 1.

Average precision is an example-based ranking metric, which evaluates the average fraction of relevant labels ranked higher than a particular label λ. It is thus given by:

$$\text{Average Precision} = \frac{1}{q}\sum_{\omega=1}^{q}\frac{1}{|\boldsymbol{y_\omega}|}\sum_{\lambda\in\boldsymbol{y_\omega}}\frac{|\{\lambda'\in\boldsymbol{y_\omega}: r_\omega(\lambda')<r_\omega(\lambda)\}|}{r_\omega(\lambda)}, \tag{10}$$

where $r(\lambda)$ is the ordered list of labels for label λ. This score corresponds to the area under precision-recall curve.

In extending a binary metric to multi-label problems, there exist a number of ways to average binary metric calculations across the set of labels. Given the True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) test samples, we calculate metrics by assuming macro- and micro-averaging across all class labels, which give equal weight for labels and instances respectively, defined as follows:

$$B_{\text{micro}} = B\left(\sum_{t=1}^{n}\text{TP}_{\lambda_t},\sum_{t=1}^{n}\text{TN}_{\lambda_t},\sum_{t=1}^{n}\text{FP}_{\lambda_t},\sum_{t=1}^{n}\text{FP}_{\lambda_t}\right),\quad B_{\text{macro}} = \frac{1}{n}\sum_{t=1}^{n}B(\text{TP}_{\lambda_t},\text{TN}_{\lambda_t},\text{FP}_{\lambda_t},\text{FN}_{\lambda_t}). \tag{11}$$

B could be any of the binary classification metrics, here the F_1 score or the Area Under the ROC Curve (AUC). In a nutshell, F-Measure conveys the balance between the precision and the recall, whereas AUC considers TP and FP rates. The bigger value obtained, the better the performance of the classifier for these metrics.

5.3 Network architecture

In order to train a deep neural network there are several hyperparameters which need to be set, including those which specify the structure of the network itself and those which determine how the network is trained. The type of the nonlinearity in the activation function is one of the first such hyperparameters that needs to be considered. We adopt the logistic sigmoid activation $\alpha_f(\phi) = \alpha_g(\phi) = \sigma(\phi) = 1/(1+e^{-\phi})$ in the hidden layers which has an output range in the interval [0,1] (and is in accordance to the the initial scaling from Eq. 7). The bias units are initialized to zero, whereas the initial weights are randomly drawn from a uniform distribution $\text{U}(-\epsilon,\epsilon)$ with $\epsilon = 4\sqrt{6/(\text{fan-in}+\text{fan-out})}$, where fan-in is the size of the previous layer and fan-out the number of hidden units in current layer[29]. Tied weights ($\boldsymbol{W_2} = \boldsymbol{W_1^\intercal}$) are commonly used to reduce the complexity, yet

[†]http://www.eea.europa.eu/data-and-maps/data/corine-land-cover-2000-raster-3

untied ($\boldsymbol{W_2} \neq \boldsymbol{W_1^\intercal}$) weights seem to generalize better in our case. Therefore, in the following results untied weights are employed in all layers.

Neural network models demand significant effort and time during training, making an exhaustive grid search in the space of hyperparameters intractable. In addition, since the particular dataset we consider has not been explored before, no prior information on where these hyperparameters approximately lie is available. As such, for the specification of the hyperparameters ρ and β which control the sparseness of the autoencoder, we first performed a coarse grid search in reasonable values and in all cases, model selection was performed according the minimum Jaccard coefficient in the validation set, which is composed of 20% of the training data (randomly sampled). More specifically, the grid is constructed by considering the set produced by the Cartesian product of $\rho \in \{0.001, 0.01, 0.1, 0.5, 0.9\}$ and $\beta \in \{1, 3, 5, 7, 9\}$ values. A more fine-grained search in the vicinity of that tuple (ρ, β) that produced the best score was subsequently considered. The models were trained for 5000 unsupervised learning epochs, while at the supervised learning stage, we use 3000 epochs with early stopping, a typical approach to prevent overfitting[29], where we monitor the validation error every 100 iterations and if it has not decreased for 500 consecutive epochs, early stopping is enabled. Reported results are averaged over 10 Monte-Carlo trials, in order to minimize the effects of the initial random seed. For the implementation of the sparse autoencoder we considered the framework described in[24] which is also available online[‡]. The optimization algorithm used for minimizing the cost function of the sparse autoencoder was the BFGS gradient descent method with limited-memory variation (L-BFGS)[§] and a stopping criterion of 10^{-8}, a quasi-Newton method for unconstrained optimization that has proved to work well.

For the implementation of RAkEL and ECC we consider the open-source MULAN[¶] Java library for multi-label learning that works on the top of WEKA[‖] data mining software. As suggested by the authors, we set the size of each labelset in RAkEL to 3 and the number of component classifiers to $2n = 40$[26], while for ECC we use 10 models[27]. SVM problem is solved with linear kernel by the Sequential Minimal Optimization (SMO) algorithm that is available within WEKA.

6. EXPERIMENTAL RESULTS

In this section, we initially investigate typically used features and their effect on our particular multi-label classification problem, serving as baseline. Subsequently, we provide a detailed performance analysis of the proposed scheme by considering two key system parameters, namely the impact of the number of neurons for a single-layer, as well as additional normalization tasks, and the impact of depth on the performance of multi-label classification algorithms, on real data. Then, we analyze the sensitivity of the feature learning model. We notice that there are also a number of other critical hyperparameters of the neural network which one can experiment on, such as the regularization parameter, the type of the nonlinearity, or even the number of units of the second hidden layer; careful selection of such parameters can potentially further improve system's performance.

6.1 Performance of raw and high quality features

In land cover classification, vegetation indices provide a stronger indicator of the amount of the photosynthetically active green biomass, than the pure spectral signatures[30]. This way, they enjoy a widespread popularity for many years. In order to obtain a clear understanding regarding the effects of the quality of the features, Table 1 presents the performance for different types of input features for both multi-label classifiers considered in this paper. In the table, the first row for each classifier case, corresponds to the optimized standardized hand-crafted features, the second to spectral reflectance values, and the third row to normalized spectral reflectance. The higher quality feature descriptors we consider are the NDVI and the Land Surface Temperature (LST) on the same tile for the same months retrieved from MODIS Terra. The combination of NDVI and LST time-series features is well established in the literature as being a very good indicator which can quantify changes in the representation of vegetation growth and physical characteristics of land cover in general[31].

[‡]http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

[§]http://www.cs.ubc.ca/ schmidtm/Software/minFunc.html

[¶]http://mulan.sourceforge.net/

[‖]http://www.cs.waikato.ac.nz/ml/weka/

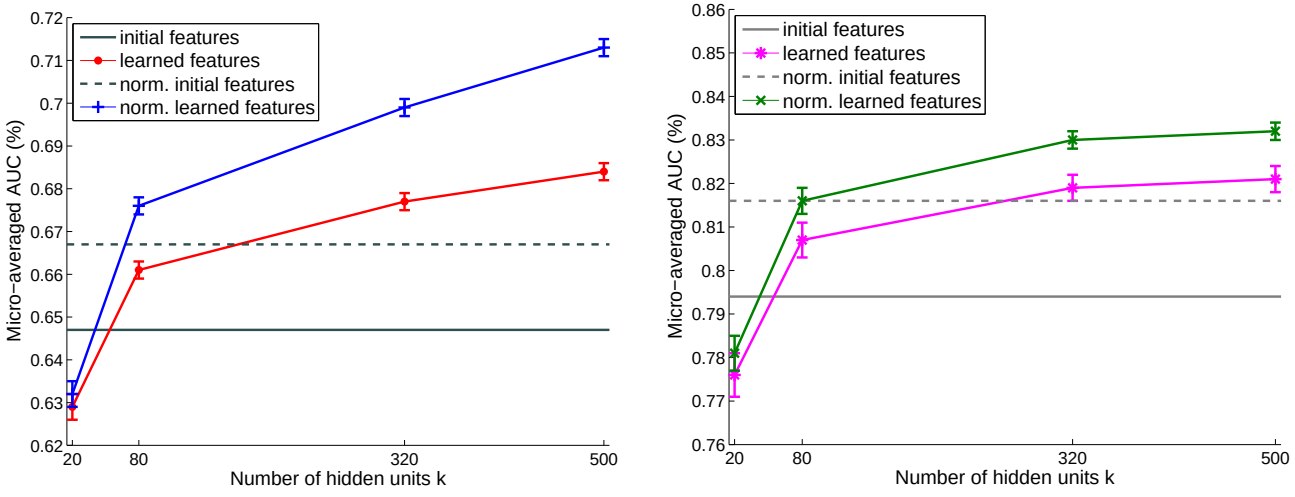
Table 1: Impact of the quality of features for the classifiers. Higher quality features composed of NDVI and LST yield to improved performance compared to raw surface reflectance.

Algorithm	Features	Hamming Loss ↓	Avg Precision ↑	Mac-F_1 ↑	Mac-AUC ↑	Mic-F_1 ↑	Mic-AUC ↑
RAkEL-SVM	NDVI–LST	0.086 ± 0.000	0.493 ± 0.000	0.224 ± 0.000	0.615 ± 0.000	0.420 ± 0.000	0.676 ± 0.000
	Surf. Refl.	0.087 ± 0.000	0.435 ± 0.000	0.157 ± 0.000	0.572 ± 0.000	0.367 ± 0.000	0.647 ± 0.000
	Norm. Surf. Refl.	0.084 ± 0.000	0.472 ± 0.000	0.175 ± 0.000	0.586 ± 0.000	0.423 ± 0.000	0.667 ± 0.000
ECC-SVM	NDVI–LST	0.087 ± 0.000	0.594 ± 0.003	0.275 ± 0.005	0.712 ± 0.006	0.474 ± 0.003	0.814 ± 0.003
	Surf. Refl.	0.087 ± 0.000	0.551 ± 0.004	0.191 ± 0.007	0.679 ± 0.008	0.449 ± 0.007	0.794 ± 0.005
	Norm. Surf. Refl.	0.085 ± 0.000	0.593 ± 0.004	0.255 ± 0.006	0.707 ± 0.005	0.486 ± 0.004	0.816 ± 0.003

The results show that the feature-level fusion of NDVI and LST acts indeed better than raw surface reflectance. This higher performance achieved by the carefully designed features is the main motivation behind our approach, which aims at formalizing an automated process able to extract more meaningful spectral characteristics from raw satellite data. Moreover, we can see that there is a significant effect of normalization on surface reflectance features, which varies from metric to metric. We emphasize than even if RAkEL and ECC are two of the most powerful schemas for multi-label classification, they perform poorly for some of the measures[32], demonstrating the dramatic challenges associated with the real-world problem we consider in this work.

6.2 Impact of layer size and normalization

We now move on to our characterization of performance on various axes of parameters, starting with the effect of layer size. Choosing the number of neurons in the hidden layers is a crucial design parameter affecting the overall neural network architecture. No formal rule guaranteeing the optimal selection exists and thus it usually comes down to trial and error. We study feature learning with both undercomplete ($k < d$) and overcomplete ($k > d$) representation models, where $d = 161$ is the dimensionality of the raw feature space, considering a single-layer architecture as a baseline. More precisely, in the case of undercomplete representations, we perform experiments with half of the initial features, $k = 80$, and then with an extreme scenario of only $k = 20$ hidden units. Correspondingly, in the case of overcomplete representations, we double the capacity of the model in breadth, $k = 320$, and then we experiment on a highly overcomplete model where the latent code has $k = 500$ units.



(a) Micro-AUC with RAkEL-SVM.

(b) Micro-AUC with ECC-SVM.

Figure 2: Performance of multi-label classifiers using the codes learned by increasing the size k of one hidden layer. The solid horizontal lines correspond to the performance of the classifiers with the initial raw features, whereas the dashed lines correspond to the accuracy achieved with the normalized version of the raw features. Optimal complexity-performance ration is achieved using twice as many of the initial raw features.

Figure 2 presents the micro-AUC score with respect to the number of hidden units used in shallow architectures, where more specifically, Figure 2a demonstrates the classification performance of the RAkEL with SVM as base classifier (RAkEL-SVM), whereas Figure 2b introduces SVM as part of an Ensemble Classifier Chain (ECC-SVM). Regarding the baselines, the solid horizontal lines correspond to the performance of the classifiers using the initial raw features, whereas the dotted lines correspond to the normalized version of the initial raw features. An observation evident in both figures is the large gap between the solid and the dotted lines suggesting that the initial normalization step of the surface reflectance data before their introduction to the classifiers can have a dramatic impact on the performance. This is in line with the fact that algorithms that work with distances and make parametric assumptions regarding the distribution of the data, such as SVM or logistic regression, are affected positively by a normalized input space in general. SVM assumes also that the data it works with lie in a standard range, thus the normalization of feature vectors is crucial. We should also notice that the computational time is much smaller with the use of normalization.

Overall, for both classifiers, 20 units are too few to adequately encode the signals in the hidden layer resulting in significant degradation performance. By increasing the number of hidden units to 80, the performance in both schemes surpasses the score achieved using raw un-normalized features as inputs. However, the gain offered by this feature learning, is outweighed to some extent by the effort of normalization of the raw input data, as indicated by the dotted line. On the contrary, by considering 320 units, the performance of the feature learning scheme increases and slightly surpasses the baseline, whereas with the incorporation of 500 units, the improvement is marginal and comes at a higher computational cost. A key observation point is that the normalization after the feature-mapping can also play a significant role and boost the performance of classifiers. In detail, we observe that in the case of the undercomplete feature learning architectures, the micro-AUC does not significantly change with or without this normalization step. Nonetheless, it is evident that in the case of overcomplete systems, the performance is higher and can clearly outperform the enhanced baseline versions with the normalized feature vectors. Note that different hyperparameters are needed in the overcomplete case whether we use normalization or not for optimal performance.

With respect to the different classifiers, one can easily notice the dominance of the ECC scheme compared to the RAkEL approach. Moreover, ECC is less affected by the normalization steps, but has a greater variance on the results. Last, we have to mention that we need the contribution of such powerful ensemble multi-label learning schemes in order to achieve reasonable performance, due to both the limited training examples and the many factors of variation that inhere in our real dataset, allowing us to test the limits of current state-of-the-art classifiers.

6.3 Impact of depth

In this set of experiments, we focus on the impact of depth, *i.e.* the number of hidden layers, with respect to the classification performance. In our setup, we employ the same number of hidden units for all layers, which has been suggested that generally leads to better performance compared to decreasing (pyramid) or increasing (inverted pyramid) network architectures[16, 29].

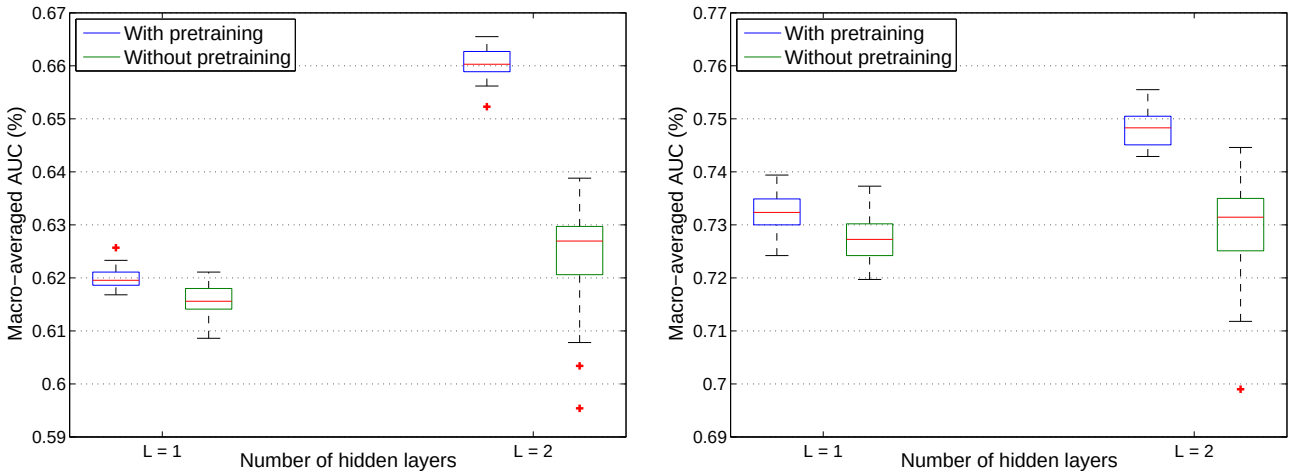
Table 2: Impact of depth for a fixed architecture consisting of 320 hidden units per layer. Higher results are obtained for features extracted from deep architectures.

Algorithm	Depth	Hamming Loss ↓	Avg Precision ↑	Mac-F_1 ↑	Mac-AUC ↑	Mic-F_1 ↑	Mic-AUC ↑
RAkEL-SVM	1	0.081 ± 0.000	0.521 ± 0.003	0.265 ± 0.004	0.620 ± 0.002	0.475 ± 0.004	0.699 ± 0.002
	2	0.082 ± 0.000	0.553 ± 0.003	0.330 ± 0.005	0.661 ± 0.003	0.504 ± 0.004	0.731 ± 0.002
	1 & 2	0.082 ± 0.000	0.568 ± 0.004	0.360 ± 0.004	0.676 ± 0.003	0.518 ± 0.003	0.743 ± 0.003
ECC-SVM	1	0.084 ± 0.000	0.623 ± 0.003	0.330 ± 0.007	0.732 ± 0.003	0.521 ± 0.003	0.830 ± 0.002
	2	0.087 ± 0.001	0.628 ± 0.003	0.377 ± 0.005	0.748 ± 0.003	0.530 ± 0.004	0.832 ± 0.002
	1 & 2	0.086 ± 0.000	0.635 ± 0.003	0.397 ± 0.005	0.757 ± 0.003	0.539 ± 0.003	0.835 ± 0.003

Table 2, provides a comprehensive numerical evaluation of the two classification schemes, namely RAkEL-SVM and ECC-SVM under different evaluation metrics. The experiments concern the features extracted from the feature learning system; either from a single-layer autoencoder (rows indicated with Depth 1), or a level-2

stacked autoencoder which obeys the properties of deep learning (rows indicated with Depth 2). The results demonstrate that both RAkEL-SVM and ECC-SVM can benefit from the additional hidden layer to gain extra valuable discriminative information. Regarding the depth of the network, the gain is significant for all metrics except hamming loss, which improves only for the first hidden layer. We noticed also that the mean value of the cost function is also smaller from the first to the second hidden layer, which can serve as a proxy of the final system's performance. In addition, we have also considered the "concatenated" representation for autoencoders (rows indicated with Depth 1 & 2 in Table 2), where we utilize the concatenation of both layers of the network. This way, the final features introduced to the classifier correspond to the combination of the first and the second hidden layer, instead of the traditional "replacement-based" representation, where only the top-layer features are used. We observe that the model can take further advantage from this kind of representation and the more features exhibiting an improved performance, but in a higher computational cost.

We have to highlight that a sparser representation has to be enforced for the second than the first hidden layer, which suggests that in this case the sparseness property in the representation can indeed help overcomplete architectures, since without the use of this type of regularization, the deep models cannot achieve performance beyond the one achieved by a single-layer architecture. Furthermore, the performance achieved with deep learning of 320 units is better compared to the single-layer case where we have 500 hidden units, further promoting the motivation for deep architectures. Finally, when the feature learning procedure is involved, the performance is substantially higher for all measures compared to the surface reflectance baselines and the higher quality features (NDVI–LST) shown in Table 1. In a nutshell, these results suggest that to really benefit from sparse overcomplete models and produce useful representations, one must consider departing from shallow to deep learning architectures.



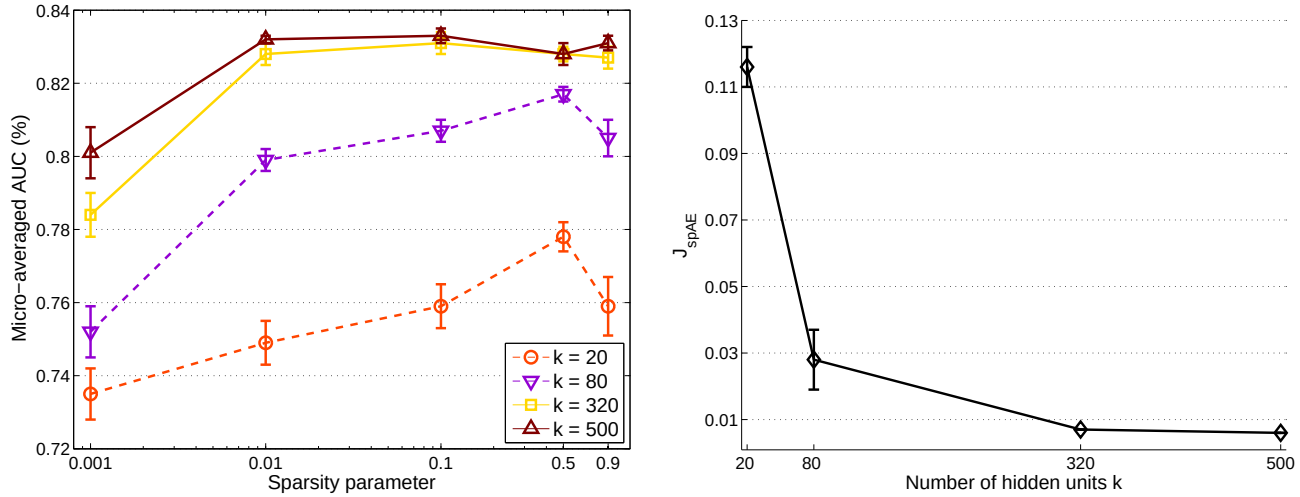
(a) RAkEL-SVM. (b) ECC-SVM.

Figure 3: Effect of depth on accuracy for models trained with and without unsupervised pretraining using RAkEL-SVM (left) and ECC-SVM (right) classifiers, for 1 to 2 hidden layers in which the hidden layer size has been fixed to 320. Box plots show the distribution of errors associated with 50 different initialization seeds. A box represents 50% of the data, the red central line indicates the median value, whereas the lower and upper boundary lines are the 25th and 75th percentiles. Whiskers extend to the remaining data that are not regarded as abnormal outliers, which are shown individually as red "+"s.

Figure 3 shows the evolution of performance as we increase the number of hidden layers from the first to the second with and without the use of pretraining via sparse autoencoders. We use the same set of hyperparameters for both models. The performance of the models with unsupervised pretraining is higher, whereas the advantage is more pronounced in deep architectures. In parallel, the pretraining procedure clearly reduces the variance of the performance, leading to more robust results. All in all, RAkEL model seems to be more affected from the pretraining procedure, as well as to benefit more from the second hidden layer in comparison to the ECC.

6.4 Model sensitivity

In this part, we investigate the sensitivity of the feature learning scheme with respect to the sparsity parameter ρ. More specifically, Figure 4a demonstrates the micro-averaged AUC score for the ECC-SVM classifier versus some representative values of the ρ, for a fixed sparsity weight ($\beta = 1$). We observe that the undercomplete models are more sensitive to the hyperparameter settings, since a change for ρ entails a dramatic change for the performance. From the other side, the overcomplete models seem more robust for different values of the sparsity parameter, but do not benefit a lot from the regularization. Overall, the hyperparameter ρ can highly affect the final performance, whereas the impact of regularization is more prominent in the undercomplete case for a single-layer architecture. In any case, as illustrated by the figure, different hyperparameters combinations can lie on a wide range, indicating that sparse autoencoders are quite sensitive models and thus hyperparameters settings have to be chosen very carefully.



(a) Micro-averaged AUC with ECC-SVM.

(b) Cost function.

Figure 4: Sensitivity of the sparse autoencoder model for the single-layer case. Sparsity parameter ρ plays an important role to the final performance for a fixed sparsity weight (left), whereas the value of the cost function reduces primarily due to the size of the hidden layer (right).

Figure 4b investigates the impact of the number of hidden units with respect to the generalization performance of sparse autoencoders as it is encoded in the cost function. We observe that the system seems to be primarily affected by the number of hidden units compared to the sparsity of the connections. By increasing the number of hidden units, the autoencoder ends-up learning a very good approximation of the identity, but the specific regularization technique does not provide much additional interpretation of the data in order to boost the performance of the subsequent classification algorithm. Intuitively, this means that for very sparse models (large value of β and small value of ρ), the algorithm tends to learn very specific features that classifiers are not capable of generalizing, thus achieving better classification rates.

7. CONCLUSIONS

The work focused on the effects that the characteristics of satellite data representation can have on a learning algorithm, an issue of extreme importance. Carefully designed hand-engineered features can significant aid in the more discriminative representation of the remote sensing data such as multispectral images employed in our case. However, the specificity of these features may limit their generalization capacity to different data sources and learning objectives. To address this issue, we propose the introduction of feature learning directly from data. Results presented in this work suggest that feature learning, in this case sparse autoencoder networks, can significantly boost the performance, even in the case of a single hidden layer. Furthermore, experiments indicate that stacking layers over the raw input data can further improve the performance leading to state-of-the-art

performance in solving a truly hard learning problem including real data that exhibits many facets of variation. Future directions include experimenting with other types of regularization, as well as extending this work to consider the nature of time-series, in order to better exploit the temporal characteristics of the features.

ACKNOWLEDGMENTS

This work was partially funded by the PHySIS project (http://www.physis-project.eu/), contract no. 640174, within the H2020 Framework Program of the European Commission.

REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**, pp. 1798–1828, Aug 2013.

[2] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Int. Conf. on Machine Learning*, *ICML '07*, pp. 473–480, ACM, (New York, NY, USA), 2007.

[3] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining* **3**(3), pp. 1–13, 2007.

[4] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Transactions on Knowledge and Data Engineering* **26**, pp. 1819–1837, Aug 2014.

[5] C. Poultney, S. Chopra, and Y. Lecun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems (NIPS)*, MIT Press, 2006.

[6] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, eds., pp. 646–654, Curran Associates, Inc., 2009.

[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.* **11**, pp. 3371–3408, Dec. 2010.

[8] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contracting auto-encoders: Explicit invariance during feature extraction," in *Int. Conf. on Machine Learning*, 2011.

[9] R. Goroshin and Y. LeCun, "Saturating auto-encoder," *CoRR* **abs/1301.3577**, 2013.

[10] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Conf. on Empirical Methods in Natural Language Processing*, *EMNLP '11*, pp. 151–161, Association for Computational Linguistics, (Stroudsburg, PA, USA), 2011.

[11] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, May 2013.

[12] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Int. Conf. on Computer Vision*, pp. 2146–2153, Sept 2009.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Int. Conf. on Artificial Intelligence and Statistics (AISTATS-11)*, G. J. Gordon and D. B. Dunson, eds., **15**, pp. 315–323, Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[14] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.* **2**, pp. 1–127, Jan. 2009.

[15] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, MIT Press, 2007.

[16] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *J. Mach. Learn. Res.* **10**, pp. 1–40, June 2009.

[17] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science* **313**(5786), pp. 504–507, 2006.

[18] R. Salakhutdinov and I. Murray, "On the quantitative analysis of deep belief networks," in *Int. Conf. on Machine Learning*, *ICML '08*, pp. 872–879, ACM, (New York, NY, USA), 2008.

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Müller, eds., *Lecture Notes in Computer Science* **7700**, pp. 9–48, Springer Berlin Heidelberg, 2012.

[20] K. Fotiadou, G. Tsagkatakis, and P. Tsakalides, "Low light image enhancement via sparse representations," in *Image Analysis and Recognition*, pp. 84–93, Springer International Publishing, 2014.

[21] G. Tsagkatakis and A. Savakis, "Sparse representations and distance learning for attribute based category recognition," in *Trends and Topics in Computer Vision*, pp. 29–42, Springer Berlin Heidelberg, 2012.

[22] P. Lennie, "The cost of cortical computation," *Current Biology* **13**(6), pp. 493 – 497, 2003.

[23] P. Petrantonakis and P. Poirazi, "A compressed sensing perspective of hippocampal function," *Frontiers in systems neuroscience* **8**, 2014.

[24] A. Ng, "Sparse autoencoder," *CS294A Lecture notes* **72**, 2011.

[25] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun, "Learning invariant features through topographic filter maps," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1605–1612, June 2009.

[26] G. Tsoumakas, I. Katakis, and L. Vlahavas, "Random k-labelsets for multilabel classification," *IEEE Transactions on Knowledge and Data Engineering* **23**, pp. 1079–1089, July 2011.

[27] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning* **85**(3), pp. 333–359, 2011.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning* **20**(3), pp. 273–297, 1995.

[29] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. Orr, and K.-R. Mller, eds., *Lecture Notes in Computer Science* **7700**, pp. 437–478, Springer Berlin Heidelberg, 2012.

[30] C. J. Tucker, "Red and photographic infrared linear combinations for monitoring vegetation," *Remote Sensing of Environment* **8**(2), pp. 127 – 150, 1979.

[31] W. P. Wan, Z. and X. Li, "Using modis land surface temperature and normalized difference vegetation index products for monitoring drought in the southern great plains, usa," *International Journal of Remote Sensing* **25**(1), pp. 61–72, 2004.

[32] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Deroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern Recognition* **45**(9), pp. 3084 – 3104, 2012.