Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

A Multi-label Cascaded Neural Network Classification Algorithm for Automatic Training and Evolution of Deep Cascaded Architecture

Arjun Pakrashi* | **Brian Mac Namee**

Insight Centre for Data Analytics, School of Computer Science, University College Dublin, Dublin, Ireland

Correspondence
*Arjun Pakrashi. Email: arjun.pakrashi@ucd.ie

Abstract

Multi-label classification algorithms deal with classification problems where a single datapoint can be classified (or labelled) with more than one class (or label) at the same time. Early multi-label approaches like binary relevance consider each label individually and train individual binary classifier models for each label. State-of-the-art algorithms like RAkEL, classifier chains, calibrated label ranking, IBLR-ML+, and BPMLL also consider the associations between labels for improved performance. Like most machine learning algorithms, however, these approaches require careful hyper-parameter tuning, a computationally expensive optimisation problem. There is a scarcity of multi-label classification algorithms that require minimal hyper-parameter tuning. This paper addresses this gap in the literature by proposing CascadeML, a multi-label classification method based on the existing cascaded neural network architecture, which also takes label associations into consideration. CascadeML grows a neural network architecture incrementally (deep as well as wide) in a two-phase process as it learns network weights using an adaptive first-order gradient descent algorithm. This omits the requirement of preselecting the number of hidden layers, nodes, activation functions, and learning rate. The performance of the CascadeML algorithm was evaluated using thirteen multi-label datasets and compared with nine existing multi-label algorithms. The results show that CascadeML achieved the best average rank over the datasets, performed better than BPMLL (one of the earliest well known multi-label specific neural network algorithms), and was similar to the state-of-the-art classifier chains and RAkEL algorithms.

KEYWORDS:
Neural Network; Machine Learning; Multi-Label

1 | INTRODUCTION

In *multi-label classification* problems a datapoint can be assigned to more than one class, or *label*, simultaneously (Herrera, Charte, Rivera, & del Jesús 2016). For example, an image can be classified as containing multiple different objects, or music can be labelled with more than one genre. This is different from *multi-class classification* problems where each object can only belong to at most one single class. Multi-label classification algorithms handle the problem in one of the two ways, as first proposed in (Tsoumakas & Katakis 2007). One way is to divide the existing multi-label problem into smaller multi-class problems—for example *classifier chains* (Read, Pfahringer, Holmes, & Frank 2011)—and this is known as *problem transformation*. The other way is to modify existing multi-class algorithms so that they can deal with the multi-label datasets—for example *BackPropagation in Multi-Label Learning* (BPMLL) (Zhang & Zhou 2006)—and this is known as *algorithm adaptation* methods.

Based on how many labels are considered together, multi-label classification algorithms can be divided into three more categories: *first-order* algorithms, *second-order* algorithms and *higher order* algorithms. First-order algorithms consider the labels independently, and therefore are not able to exploit associations that exist between labels (inter-label associations). Second-order algorithms consider labels in a pairwise manner, for example by considering the relative ordering or the interaction between two labels at a time. Higher order algorithms consider relations between more than two labels at the same time. For example a higher order algorithm can find how a set of labels influence a specific label, and use that to improve label predictions. Several studies find that algorithms which take some consideration of label associations perform better than the first-order algorithms (Madjarov, Kocev, Gjorgjevikj, & Džeroski 2012; Pakrashi, Greene, & Mac Namee 2016). A comprehensive review of the recent ongoing research in multi-label learning can be found in (Gibaja & Ventura 2014).

Neural networks are very effective in multi-class classification, but have not been extensively researched in the domain of multi-label classification. Some research has been done with neural networks, which solves multi-label problems like in (Chen, Chi, Fu, & Feng 2013; Read & Perez-Cruz 2014; Wei et al. 2014; Yu, Wang, Zhang, Gong, & Zhao 2017; Zhu, Liao, Lei, & Li 2017; Zhuang, Yan, Chen, Wang, & Shen 2018), but they do not explicitly consider inter-label associations. Instead, they either use neural network as a part of the processing pipeline or weights the influence of the labels manually. *BPMLL* (Zhang & Zhou 2006) was the first proposed neural network algorithm which takes label associations into consideration.

An essential part in building a machine learning model with good generalisation performance is hyper-parameter tuning. Multi-label classification approaches also face the same challenge as any other machine learning algorithms. The task of selecting the best hyper-parameter setting for an algorithm is an optimisation problem by itself. Very limited work has been done on automatic hyper-parameter tuning and AutoML (Feurer et al. 2015) in the multi-label domain.

Motivated by *cascade correlation neural networks* (Fahlman & Lebiere 1990), and BPMLL (Zhang & Zhou 2006), the work presented in this paper attempts to fill this gap by proposing a neural network algorithm, CascadeML, to train multi-label neural networks based on the cascade neural network architecture. This method requires no hyper-parameter tuning and also considers inter-label associations. The cascade algorithm grows the network architecture incrementally in a two phase process as it learns the weights using an adaptive first-order gradient algorithm. Except for setting some upper and lower limits, CascadeML omits the requirement of specifically preselecting the number of hidden layers and nodes in the network and the learning rate, therefore making the training process automatic.

Through a series of extensive evaluation experiments, when compared to several state-of-the-art multi-label classification algorithms, CascadeML was found to perform very well without a requirement of explicit hyper-parameter tuning and out-perform several state-of-the-art methods over several benchmark datasets. To the best of authors' knowledge this is the first automatic neural network architecture selection and training approach for multi-label classification methods.

The paper is structured as follows. Section 2 discusses the related work, Section 3 introduces the proposed method. Section 4 explains the experimental design and the results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 | RELATED WORK

The *cascade correlation neural network* (Fahlman & Lebiere 1990) and *Cascade2* algorithm (Prechelt 1997) are interesting neural network approaches that learn model parameters and model architecture at the same time. In cascade correlation neural network and Cascade2 approaches, training starts with a simple perceptron network, which is grown incrementally by adding new single *cascade units* with skip-level connections as long as performance on a validation dataset improves. Each cascade units accepts incoming weights from the inputs and also from all previous hidden cascade layers, and has outgoing weight connections to the output units. Weights in each new layer are trained independently of the overall network which greatly reduces the processing burden of this approach. Since the proposal of the original cascade correlation algorithm in (Fahlman & Lebiere 1990), a few improvements that follow a similar overall process to the original method have been proposed, for example in (Baluja & Fahlman 1994; Hansen & Pedersen 1994; Phatak & Koren 1994; Waugh & Adams 1994). Active research in this field, however, is fairly limited.

Automatic machine learning (Feurer et al. 2015), or *AutoML*, approaches have seen a recent resurgence of interest as researchers look for ways to automatically select optimal algorithms, features, model architectures, and hyper-parameters for machine learning tasks. The AutoML research community has, however, paid very little attention to the multi-label classification domain, although there have been some recent efforts (de Sá, Freitas, & Pappa 2018; de Sá, Pappa, & Freitas 2017; Wever, Mohr, & Hüllermeier 2018).

Back Propagation in Multi-Label Learning (BPMLL), proposed by Zhang et al. in 2006 (Zhang & Zhou 2006), is a single hidden layer, fully connected feed-forward neural network architecture for multi-label classification. BPMLL uses the back-propagation (Haykin 1998) algorithm to optimise a variation of the ranking loss function (Zhang & Zhou 2014) that takes inter-label associations into account. The loss function minimised by BPMLL is defined as follows:

$$E = \sum_{i=1}^{n} \frac{1}{|\mathbf{y}_i||\bar{\mathbf{y}}_i|} \sum_{(k,l)\in(\mathbf{y}_i\times\bar{\mathbf{y}}_i)} exp\left(-(c_i^{(k)} - c_i^{(l)})\right) \tag{1}$$

Here $\mathbf{y}_i$ indicates the set of labels relevant to $\mathbf{x}_i$ and $\bar{\mathbf{y}}_i$ indicates the set of label which are not relevant to $\mathbf{x}_i$. The network uses the tanh activation function, therefore this algorithm uses a bipolar encoding of the target variables: $y_i^{(l)} = +1$ if the label l is relevant to $\mathbf{x}_i$, and -1 if irrelevant. Here $c_i^{(k)}$ and $c_i^{(l)}$ are the outputs of the k^{th} and the l^{th} output units representing the corresponding label predictions for the datapoint $\mathbf{x}_i$.

The CasecadeML algorithm uses elements of the original cascade correlation algorithm and the BPMLL loss function and is described in detail in the next section.

TABLE 1 BPMLL cost function behaviour

Label assignment	$c_i^{(k)}$	$c_i^{(l)}$	$-(c_i^{(k)} - c_i^{(l)})$	$\exp^{-(c_i^{(k)} - c_i^{(l)})}$
Wrong	1.00	1.00	0.00	1.0000
Correct	**1.00**	**-1.00**	**-2.00**	**0.1353**
Wrong	-1.00	1.00	2.00	7.3891
Wrong	-1.00	-1.00	0.00	1.0000

The intuition behind this loss function is that for a pair of labels $(k, l) \in (\mathbf{y}_i \times \bar{\mathbf{y}}_i)$, where k is relevant to the datapoint $\mathbf{x}_i$ and l is not, if the prediction score for k is positive whereas the prediction score for l is negative, then $\exp(-(c_i^{(k)} - c_i^{(l)}))$ has the minimum penalty. An incorrect prediction score order results in higher penalty. Therefore, minimising Eq. (1) would result in pairs of labels being predicted correctly. Table 1 shows an example. If $c_i^{(k)}$ is predicted as $+1$ and $c_i^{(l)}$ is predicted as -1 (boldface the Table 1), then the predictions of the network is correct, which has the lowest penalty, whereas any other combinations are heavily penalised[1].

3 | THE CASCADEML ALGORITHM

CascadeML is a cascaded neural network approach to multi-label classification motivated by on cascade correlation neural networks (Fahlman & Lebiere 1990) and Cascade2 (Prechelt 1997). The main objective of this method is to find good multi-label classifier models that take advantage of label associations, while minimising the model selection and training time by minimising or altogether omitting hyper-parameter tuning and architecture tuning. This section describes the proposed CascadeML algorithm. The generic architecture of a cascade neural network is first described, before the specific training method for CascadeML is presented.

3.1 | Architecture

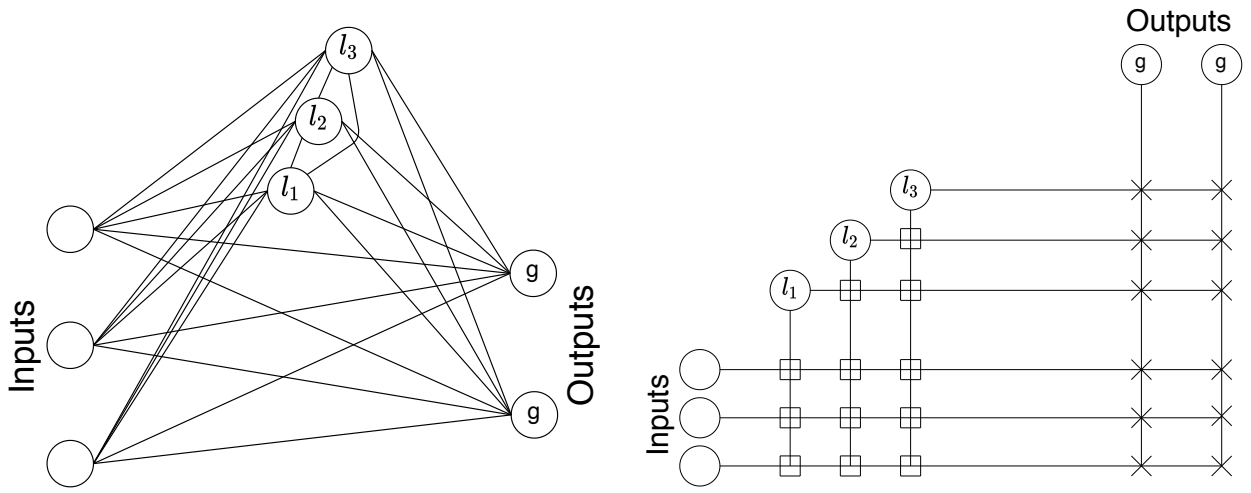
The network architecture trained using CascadeML has $d + 1$ inputs (including a bias term) and q outputs (one for each label). Bipolar (Haykin 1998) encoding is used for the outputs such that Eq. (1) can be used, in which a relevant label is represented using $+1$ and an irrelevant label is represented using -1. Each of the network's L hidden layers, l_i, can have any number of units, which receives incoming weights from all the $d + 1$ inputs as well as from all the hidden units in the previous layers. The output of each hidden layer l_i is connected to the q outputs of the network. A layer with such a connection scheme is called a *cascade layer*.

The weights of a cascade network can be divided into four categories

1. **Input to output layer weights** connecting the $d + 1$ inputs to the q outputs, forming a perceptron network.
2. **Input to hidden layer weights** connecting the $d + 1$ inputs to the L hidden cascade layers.
3. **Hidden to hidden layer weights** connecting the output of all the previous hidden cascade layers $l_1, l_2, \ldots, l_{i-1}$, to the hidden cascade layer l_i.
4. **Hidden to output layer weights** connecting the outputs of the cascade layers $l_1, l_2, \ldots, l_L$ to the q output units.

Figure 1 shows an example of a simple cascade neural network with three inputs, two output labels, and three hidden cascade layers (l_1, l_2, and l_3). All connections flow from left to right. The architecture of CascadeML has the same architecture as cascaded networks, although they are

[1]The values $+1$ and -1 are used to indicate the upper and lower bound of the tanh activation function.



(a) Final trained architecture, equivalent to Fig. 2g.

(b) Phase I, cascade layer 3 added, training output weights, input to hidden and hidden to hidden weights frozen.

FIGURE 1 Cascade network architecture. Both Figure 1a and Figure 1b are different representations of the same architecture, where 1b is used in (Fahlman & Lebiere 1990).

trained differently. Figure 1b shows the same network using an alternate representation of Figure 1a defined in (Fahlman & Lebiere 1990). This representation will be used in the next section to explain the algorithm.

A cascade network starts from a simple perception network and then grows dynamically one layer at a time. Each layer has a cascaded connection as described previously. The different types of weights are trained in an iterative two-phase process, in the next section. Once the training is complete, prediction uses a general feed-forward algorithm that propagates the inputs through the cascade layers.

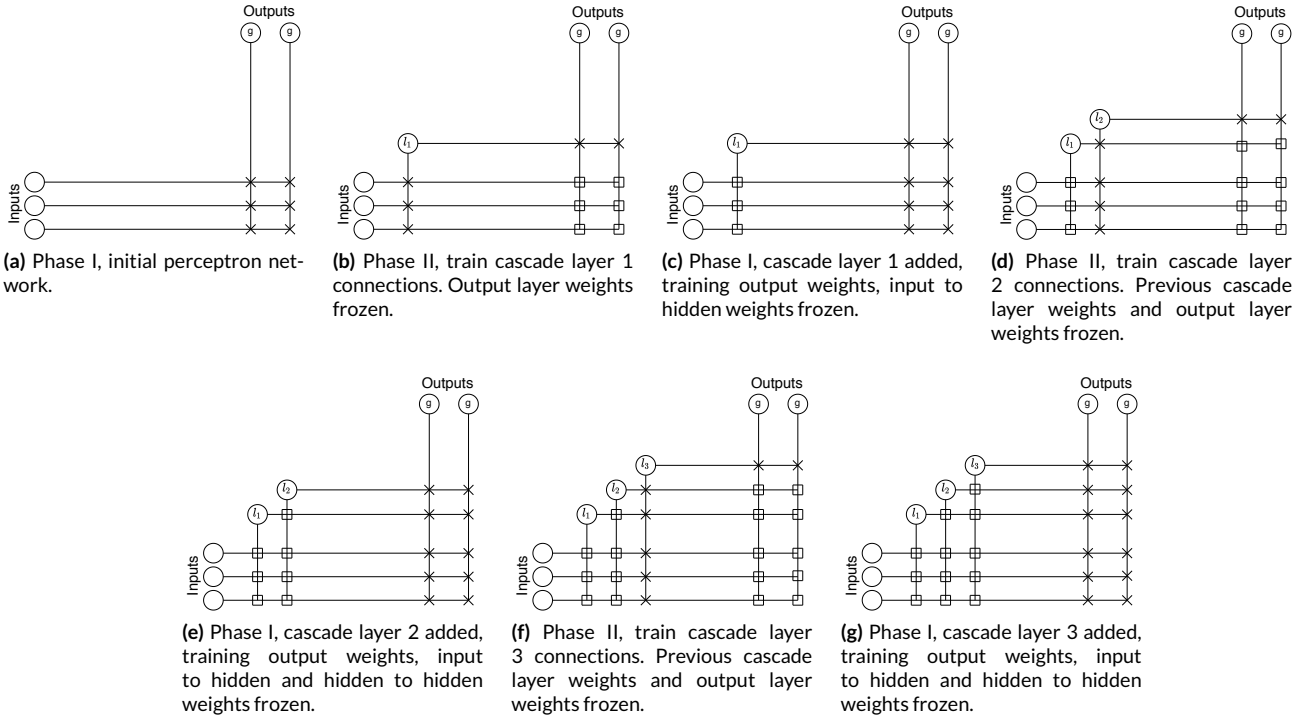
3.2 | Training

Model training in the CascadeML algorithm starts with a simple perceptron network (Figure 2a) with $d + 1$ inputs (including the bias unit) and q output units, one for each label. This network is referred to as the *main network*. The main network is grown as training proceeds by iteratively adding hidden cascade layers to it. This is achieved by iteratively repeating two phases, *Phase I* and *Phase II*, each of which trains different parts of the cascade network independently.

In Phase I, the input to output layer weights (type 1, as mentioned in Section 3.1) and hidden to output layer weights (type 4) of the main network are trained, while all other weights (input to hidden layer and hidden to hidden layer) are frozen. The target values used in this phase to calculate the loss of the network are the target classes from the original dataset. The BPMLL cost function described in Eq.(1) between the output of the main network and the ground truth is minimised using gradient descent.

Phase II trains and adds a new cascade layer, l_i, at the i^{th} iteration of training. The inputs to the newly added layer, l_i, are the $d + 1$ inputs, and the outputs from the previous hidden layers $l_1, \ldots, l_{i-1}$, in the main network. At this phase only the weights related to the new hidden layer, l_i, are trained. These are the input to hidden layer weights (type 2) for l_i; hidden to hidden layer weights on connections of the output of previous hidden cascade layers, $l_1, \ldots, l_{i-1}$, to the current hidden layer, l_i (type 3); and the weights connecting the new hidden layer, l_i, to the output layer (type 4). All other weights in the main network are frozen. In this phase the target values used in training are not the original target values, but rather the error between the MSE (*Mean Squared Error*) of the main network constructed up to the previous iteration $i - 1$, and the output of the new layer l_i. Therefore this phase makes the new cascaded layer fit the MSE of the main network.

When adding a new cascade layer the number of nodes to include, the type of activation function to use and other hyper-parameters need to be determined. These give rise to a hyper-parameter selection problem. To overcome this, at each iteration of CascadeML, a *candidate pool* of many candidate hidden cascade layers that could be added to the main network is trained. Each of the candidate hidden cascade layer is initialised with 1) randomly selected initial weight values, 2) a randomly selected activation function, and 3) a randomly selected number of units. Each of the candidate hidden cascade layers in the candidate pool is trained independently in parallel, to minimise MSE as explained before. Once they have all been trained for every candidate network in the candidate pool, the best candidate hidden cascade layer from the candidate pool is selected (based



(a) Phase I, initial perceptron network.

(b) Phase II, train cascade layer 1 connections. Output layer weights frozen.

(c) Phase I, cascade layer 1 added, training output weights, input to hidden weights frozen.

(d) Phase II, train cascade layer 2 connections. Previous cascade layer weights and output layer weights frozen.

(e) Phase I, cascade layer 2 added, training output weights, input to hidden and hidden to hidden weights frozen.

(f) Phase II, train cascade layer 3 connections. Previous cascade layer weights and output layer weights frozen.

(g) Phase I, cascade layer 3 added, training output weights, input to hidden and hidden to hidden weights frozen.

FIGURE 2 The steps in the Cascade2 network training algorithm. In each diagram the circles labelled *Inputs* correspond to the input layer and the circles labelled *Outputs* correspond to the output layer of the network. Hidden cascade units are represented by the circles labelled l_i. A weight between nodes in two layers exists, where horizontal and vertical lines intersect. Crosses indicate a weight that is trainable at a specific step in the training process, while squares indicate a weight that is frozen at a specific step. g is the activation function used at the output layer.

on calculated loss on a validation dataset) and added to the main network. The algorithm includes candidate hidden cascade layers that are *sibling* layers to the deepest hidden cascade layer already in the main network (Baluja & Fahlman 1994), as well as *successor* cascade layers. If the new trained candidate layer is added as a sibling to the existing deepest layer, then this existing deepest layer expands in width, that is, the nodes in the deepest layer increases. On the other hand the candidate layer added as a successor increases the depth of the network by adding this layer to a new cascade layer in the main network. This allows wide architectures as well as deep architectures to be explored. This is done by training candidate cascade networks in the candidate pool as successors and siblings and then selecting the best of the two types of candidate network.

Once the weights associated with the new hidden layer have been trained the layer and these weights are added to the main network. The weights connecting the new hidden layer, l_i, to the output layer are negated when these are added to the main network. This is so that the contribution of the output of the newly added layer will minimise the error of the main network (Nissen 2007)—recall that the newly added layer was trained to predict the main network error (a regression problem), and not the target labels.

The candidate hidden cascade layers in the candidate pool can each be trained independently in isolation from the main network, because when training the candidate hidden cascade layer, l_i, the inputs to the layer, the targets and the weights of the main network are all fixed. Therefore, the hidden cascade layer, l_i, can be considered a subnetwork, trained in isolation and then added to the main network. As the candidate networks in the candidate pool can be trained in isolation, therefore all the candidate networks in the pool can be trained in parallel.

Training in both Phase I and Phase II uses a gradient based algorithm. Because Phase I effectively trains a perception network and Phase II trains a single hidden layer network, fast gradient algorithms are used. In the original cascade correlation network training algorithm, Quickprop was used (Fahlman 1988). Initial experiments uncovered that Quickprop was unstable for this specific application and therefore an adaptive first-order gradient descent algorithm iRProp- (Igel & Hüsken 2000), a variant of RProp (Igel & Hüsken 2000; Rojas 1996), is used in the CascadeML training process. iRProp- was found to be more stable than the originally used Quickprop (Fahlman 1988). iRProp- is an adaptive algorithm which uses an adaptive learning rate and the sign of the partial derivative of the error function for each weight adjustment. This method mainly helps learn very fast in the flat regions and near local minima of the error surface of the cost function, as it uses only the sign of the partial derivative (ignoring its magnitude) and uses an adaptive learning rate. L2 regularisation (Goodfellow, Bengio, & Courville 2016) was used in both phases of CascadeML.

TABLE 2 Multi-label datasets

Dataset	Instances	Inputs	Labels	Labelsets	Cardinality	MeanIR
flags	194	19	7	24	3.392	2.255
yeast	2417	103	14	77	4.237	7.197
scene	2407	294	6	3	1.074	1.254
emotions	593	72	6	4	1.869	1.478
medical	978	1449	45	33	1.245	89.501
enron	1702	1001	53	573	3.378	73.953
birds	322	260	20	55	1.503	13.004
genbase	662	1186	27	10	1.252	37.315
cal500	502	68	174	502	26.044	20.578
llog	1460	1004	75	189	1.180	39.267
Water-quality	1060	16	14	852	5.073	1.767
foodtruck	407	21	12	116	2.290	7.094
PlantPseAAC	978	440	12	32	1.079	6.690

When Phase II is complete, the algorithm proceeds again to Phase I and continues iterating between Phase I and Phase II until a maximum depth is reached or a learning error threshold is not exceeded. Training always ends with Phase I.

3.3 | Example

Figure 2 shows an example of the growth of a cascade network (the neural network diagram scheme used by (Fahlman & Lebiere 1990) is used). Figure 2a shows the initial network with 3 inputs and 2 outputs. In this schematic the intersections of the straight lines indicate the weights. A cross at an intersection indicates a trainable weight at the current phase, while a square indicates a frozen weight. The algorithm starts in Phase I and the network in Figure 2a is trained. All input to output layer weights (type 1), are trained (no hidden to output layer weights (type 4) exist yet). Next, in Phase II, a new cascade layer, I_1, is added as shown in Figure 2b, and only the input to hidden layer (type 2) and hidden to hidden layer (type 3) weights related to the newly added layer, I_1, are trained. Next, the process goes back to Phase I and trains input to output layer (type 1) and hidden to output layer (type 4) weights in the main network as shown in Figure 2c. This process iterates two more times through Phase I and II in Figures 2d, 2e and 2f until the final network in Figure 2g is produced.

4 | EXPERIMENT DESIGN

An extensive set of experiments was performed to understand the effectiveness of CascadeML where its performance was compared with nine well-known multi-label classification algorithms over thirteen multi-label datasets[2]. The datasets used are described in Table 2, where *Instances*, *Inputs* and *Labels* are the number of datapoints, the dimension of the dataset, and the number of labels respectively; *Labelsets* indicates the number of unique label combinations present; *Cardinality* measures the average number of labels assigned to each datapoint; and *MeanIR* (Charte, Rivera, del Jesus, & Herrera 2014) indicates the imbalance ratio of the labels.

The performance of models trained using CascadeML was compared with nine other multi-label and state-of-the-art multi-label classification algorithms: BPMLL (Zhang & Zhou 2006), *classifier chains* (Read et al. 2011) (CC), *RAkEL* (Tsoumakas & Vlahavas 2007), *calibrated label ranking* (CLR) (Fürnkranz, Hüllermeier, Mencía, & Brinker 2008), HOMER (Tsoumakas, Katakis, & Vlahavas 2008), MLkNN (Zhang & Zhou 2007), IBLR-ML+ (Cheng & Hüllermeier 2009), *binary relevance* (BR) (Boutell, Luo, Shen, & Brown 2004) and *label powerset* (LP) (Boutell et al. 2004). These algorithms were selected to cover different types of multi-label classification techniques. BPMLL is a well-known multi-label specific neural network algorithm; Classifier chains, RAkEL and HOMER are ensemble methods that have been shown to achieve state-of-the-art performance (Madjarov et al. 2012; Pakrashi et al. 2016); MLkNN is a nearest-neighbour based algorithm adaptation method and IBLR-ML+ is an improved nearest-neighbour based algorithm adaptation method which takes label associations into consideration. The early and simple problem transformation algorithms BR and

[2]Datasets are available at: http://www.uco.es/kdis/mllresources/ and http://mulan.sourceforge.net/datasets-mlc.html

LP were also compared as they can perform well when tuned properly (Grodzicki, Mańdziuk, & Wang 2008; Pakrashi et al. 2016). The implementations of these existing algorithms are from the MULAN library (Tsoumakas, Spyromitros-Xioufis, Vilcek, & Vlahavas 2011) and implemented in Java. CascadeML was implemented in Python [3].

The label-based macro-averaged F-Score (Zhang & Zhou 2014) was used to compare the prediction performances of the different algorithms. This was preferred over Hamming loss (Zhang & Zhou 2014), which is often used (Cheng & Hullermeier 2009; Spyromitros, Tsoumakas, & Vlahavas 2008; Zhang & Zhou 2007), because of the implicitly highly imbalanced nature of the label space. In highly imbalanced multi-label datasets Hamming loss tends to allow the performance on the majority labels to overwhelm performance on the minority labels. Label-based macro-averaged F-Score does not suffer from this problem. For every dataset performance is evaluated using a 2 times 5-fold cross-validation experiment. The mean label-based macro-averaged F-Score from these experiments are reported. The results presented are based on the best performing hyper-parameter combinations found through tuning for all algorithms except CascadeML, which does not require explicit hyper-parameter tuning.

4.1 | Configuring CascadeML

Although there is no hyper-parameter tuning required for CascadeML, it does require some configuration. In the experiments described here, at each iteration, the candidate pool contained two candidates for each combination of layer type—successor or sibling—and activation unit type—linear, sigmoid, or tanh. Thus, a total of 12 candidate hidden cascade layers are trained at each iteration in parallel. The number of hidden units in each candidate layer was randomly selected as a fraction of the number of input dimensions, d, following a uniform distribution in $(0, 1]$.

Note that the activation function for the output of the main network and the outputs of the candidate networks are always fixed, as per the requirement of the algorithm. The activation function used at the output layer of the main network was tanh, as the cost function Eq. (1) requires bipolar encoding of the labels. During Phase II of training the outputs of the candidate layers in the pool use a linear activation function, as explained in Section 3.2. The activation functions of the hidden units of the cascaded layers are selected from the set $\{\text{linear}, \text{sigmoid}, \text{tanh}\}$, and the best configuration is selected from the candidate pool.

A fast and adaptive first-order gradient descent algorithm, iRProp- was used and is initialised as recommended in (Igel & Hüsken 2000). The maximum number of allowed iterations is kept to 2000, although an early stopping mechanism is used. The training stops if the average loss, based on a validation dataset, calculated over a window of the last 20 training cascade training epochs (Phase I and Phase II iterations) increases from one iteration to the next. L2 regularisation was used in both the training phases with a regularisation value of 10^{-5}. This caps the number of selection and addition of a cascaded network (added as a sibling or successor) to 20.

4.2 | Configuring Other Algorithms

The other algorithms used in the experiment used a grid search hyper-parameter tuning using 2 time 5-folds cross-validation. For classifier chains, RAkEL, HOMER, calibrated label ranking, binary relevance and label powerset, support vector machines (Kelleher, Mac Namee, & D'Arcy 2015) with a radial basis kernel (SVM-RBF) were used as the base classifier. In these cases 12 combinations of the regularisation parameter, C, and the kernel spread, σ, were included the hyper-parameter grid. For RAkEL the subset size hyper-parameter (ranging from 2 to 6) was included, and for HOMER the cluster size hyper-parameter (ranging from 2 to 6) was also included. For MLkNN and IBLR-ML+, the value of the number of nearest neighbours k was tuned between 4 to 26 with a step size of 2. For BPMLL the only hyper-parameter in the grid search was the number of units in the hidden layer. Sizes of 20%, 40%, 60%, 80% and 100% of the number of inputs for each dataset were explored, as recommended by (Zhang & Zhou 2006). In this case the L2 regularisation coefficient was set to 10^{-5} and a maximum of 10000 iterations were allowed, based on (Pakrashi et al. 2016).

5 | RESULTS

The analysis of the results will be done in three phases. Firstly, the model performance from the cross-validation experiments will be directly compared in Section 5.1. Next, statistical significance tests will be performed to understand the differences between the algorithms and CascadeML in Section 5.2. Finally, some properties of the automatically generated deep cascaded architecture will be investigated in Section 5.3.

[3] A version of CascadeML is available at: https://github.com/phoxis/CascadeML

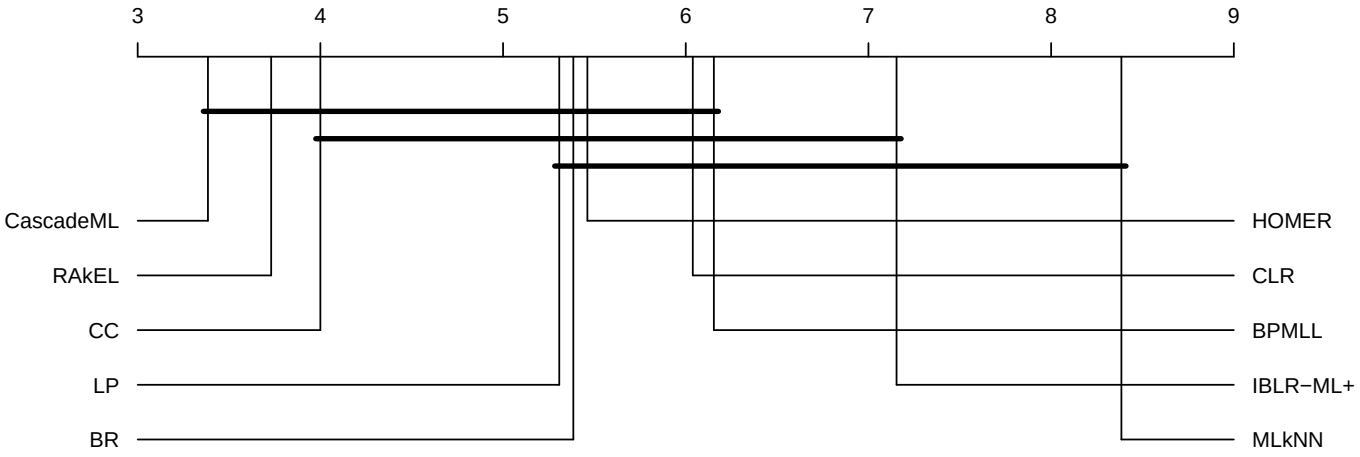


FIGURE 3 Rank plot of Friedman rank sum test with Finner p-value correction. The scale indicates the average ranks. The methods which are not connected with the horizontal lines are significantly different with a significance level of 0.05.

5.1 | Algorithm Performance

The results of the experiments are shown in Table 3, where the columns indicate the algorithms and the rows indicate the datasets. Each cell of the table shows the label-based macro-averaged F-Score (higher values are better) followed by the standard deviation over the cross validation folds. The values in the parentheses indicate the relative ranking (lower values are better) of the algorithm with respect to the other algoirithms for each dataset. The last row of the Table 3 indicates the average rank of each algorithm. The algorithms are sorted from best to the worst average rankings in Table 3.

Table 3 shows that CascadeML (avg. rank 3.38) performed the best based on the average ranks. The performance of CascadeML was a lot better than BPMLL (avg. rank 6.15), HOMER (avg. rank 5.46), MLkNN (avg. rank 8.38), LP (avg. rank 5.31), BR (avg. rank 5.38) and CLR (avg. rank 6.04). Among the other state-of-the-art multi-label classification algorithms CascadeML performed better than RAkEL (avg. rank 3.73) and CC (avg. rank 4.00) but they were very close. Although RAkEL and CC had relatively close average ranks to CascadeML, this was only achieved after extensive hyper-parameter tuning which CascadeML did not require. Absolute running times or the number of operations performed by each algorithm are not directly comparable between the methods used as the methods are so different and implementations of the algorithms span different programming languages. However, it is worth noting that completing the benchmark for CC and RAkEL (and many of the other algorithms) took multiple weeks each (with multiple folds run in parallel) due to the hyper-parameter tuning involved, whereas running the equivalent benchmark for CascadeML took less than one week.

5.2 | Statistical Significance Testing

To further analyse overall differences between the algorithm performances over the datasets selected for this experiment statistical significance tests were performed.

5.2.1 | Multiple Classifier Comparison

To understand the overall differences of the algorithms a multiple classifier comparison test, Friedman rank test (García, Fernández, Luengo, & Herrera 2010) was performed with the Finner p-value correction. The results of this evaluation are summarised in Figure 3, where the scale indicates the average ranks. If the methods are not connected with a horizontal line then they are significantly different over different datasets with a significance level of 0.05. From this test it can be seen that overall CascadeML performed significantly better than MLkNN and IBLR-ML+. Although for the other algorithms, the null hypothesis could not be rejected with the significance level $\alpha = 0.05$. Although, from the rank chart in Figure 3 it can be seen that CascadeML, RAkEL and CC was close in ranking, although CascadeML achieved the best rank.

5.2.2 | Isolated Pairwise Comparison

To further understand how individual algorithm pairs compare with each other, ignoring other algorithms, a two tailed Wilcoxon's Signed Rank Sum test for each pair of algorithms was performed. The cells of Table 4 in the lower diagonal show the p-values of the Wilcoxon's Signed Rank Sum

TABLE 3 Results of experiments. Rows indicate the datasets, columns indicate algorithms. Values in cells are mean label-based macro-averaged F-Scores and the standard deviations over 2 times 5-folds cross-validation followed by relative ranks in parentheses.

	CascadeML	RAkEL	CC	LP	BR	HOMER	CLR	BPMLL	IBLR-ML+	MLkNN
flags	0.6723 ± 0.06 (1)	0.6307 ± 0.05 (7)	0.6405 ± 0.06 (6)	0.6193 ± 0.04 (8)	0.6570 ± 0.04 (4)	0.6479 ± 0.04 (5)	0.6636 ± 0.05 (2)	0.5948 ± 0.03 (10)	0.6623 ± 0.04 (3)	0.6009 ± 0.07 (9)
yeast	0.4624 ± 0.01 (1)	0.4367 ± 0.02 (5)	0.4510 ± 0.01 (3)	0.4511 ± 0.01 (2)	0.3866 ± 0.01 (9)	0.4478 ± 0.02 (4)	0.3989 ± 0.01 (8)	0.4357 ± 0.01 (6)	0.4326 ± 0.01 (7)	0.3772 ± 0.01 (10)
scene	0.7606 ± 0.01 (8)	0.8017 ± 0.01 (3)	0.8040 ± 0.01 (1)	0.8020 ± 0.01 (2)	0.7988 ± 0.01 (5)	0.8001 ± 0.02 (4)	0.7939 ± 0.01 (6)	0.7777 ± 0.01 (7)	0.6555 ± 0.01 (10)	0.7424 ± 0.02 (9)
emotions	0.6671 ± 0.02 (2)	0.6281 ± 0.02 (4)	0.6242 ± 0.01 (5)	0.5960 ± 0.02 (10)	0.6043 ± 0.02 (9)	0.6212 ± 0.02 (7)	0.6157 ± 0.03 (8)	0.6899 ± 0.02 (1)	0.6240 ± 0.02 (6)	0.6294 ± 0.03 (3)
medical	0.6758 ± 0.02 (4)	0.6966 ± 0.03 (1)	0.6924 ± 0.04 (2)	0.6589 ± 0.06 (5)	0.6759 ± 0.04 (3)	0.6108 ± 0.05 (6)	0.5204 ± 0.20 (10)	0.5582 ± 0.08 (8)	0.5998 ± 0.04 (7)	0.5398 ± 0.05 (9)
enron	0.2852 ± 0.02 (4)	0.2882 ± 0.04 (2)	0.2890 ± 0.03 (1)	0.2784 ± 0.04 (8)	0.2842 ± 0.04 (5)	0.2812 ± 0.03 (6)	0.2862 ± 0.04 (3)	0.2806 ± 0.02 (7)	0.1965 ± 0.02 (9)	0.1771 ± 0.03 (10)
birds	0.4812 ± 0.03 (1)	0.1812 ± 0.06 (5)	0.1582 ± 0.06 (7)	0.1807 ± 0.09 (6)	0.1568 ± 0.07 (8)	0.1551 ± 0.05 (10)	0.1562 ± 0.07 (9)	0.3426 ± 0.06 (2)	0.2558 ± 0.03 (3)	0.2256 ± 0.09 (4)
genbase	0.8301 ± 0.04 (9)	0.9432 ± 0.05 (2)	0.9440 ± 0.04 (1)	0.9408 ± 0.03 (4)	0.9414 ± 0.07 (3)	0.9394 ± 0.05 (5)	0.9312 ± 0.04 (7)	0.8149 ± 0.12 (10)	0.9392 ± 0.04 (6)	0.8502 ± 0.05 (8)
cal500	0.2263 ± 0.01 (2)	0.1790 ± 0.01 (7)	0.1849 ± 0.01 (4)	0.1775 ± 0.01 (8)	0.1805 ± 0.01 (6)	0.1988 ± 0.02 (3)	0.1685 ± 0.04 (9)	0.2367 ± 0.02 (1)	0.1827 ± 0.01 (5)	0.1007 ± 0.01 (10)
llog	0.2264 ± 0.02 (9)	0.2998 ± 0.05 (1)	0.2916 ± 0.03 (5)	0.2965 ± 0.02 (2)	0.2962 ± 0.04 (3)	0.2561 ± 0.03 (8)	0.2812 ± 0.04 (6)	0.2953 ± 0.06 (4)	0.2239 ± 0.04 (10)	0.2630 ± 0.05 (7)
Water-quality	0.5726 ± 0.01 (1)	0.5212 ± 0.02 (3.5)	0.5068 ± 0.02 (7)	0.5132 ± 0.01 (5)	0.5094 ± 0.01 (6)	0.5598 ± 0.01 (2)	0.5212 ± 0.01 (3.5)	0.4916 ± 0.02 (8)	0.4819 ± 0.01 (9)	0.4151 ± 0.02 (10)
foodtruck	0.2809 ± 0.05 (1)	0.2807 ± 0.03 (2)	0.2798 ± 0.01 (3)	0.2636 ± 0.03 (6)	0.2792 ± 0.03 (4)	0.2560 ± 0.03 (7)	0.2774 ± 0.03 (5)	0.2282 ± 0.02 (8)	0.1941 ± 0.03 (9)	0.1177 ± 0.04 (10)
PlantPseAAC	0.2955 ± 0.04 (1)	0.2157 ± 0.03 (6)	0.2109 ± 0.03 (7)	0.2295 ± 0.04 (3)	0.2163 ± 0.02 (5)	0.2270 ± 0.04 (4)	0.2342 ± 0.03 (2)	0.1925 ± 0.03 (8)	0.1866 ± 0.02 (9)	0.0505 ± 0.03 (10)
Avg. rank	3.38	3.73	4.00	5.31	5.38	5.46	6.04	6.15	7.15	8.38

TABLE 4 Upper diagonal: win/lose/tie. Lower diagonal: Results of the Wilcoxon's Signed Rank test. * $\alpha = 0.1$, ** $\alpha = 0.05$ and *** $\alpha = 0.01$

	CascadeML	RAkEL	CC	LP	BR	HOMER	CLR	BPMLL	IBLR-ML+	MLkNN
CascadeML		8/5/0	8/5/0	10/3/0	9/4/0	10/3/0	9/4/0	9/4/0	12/1/0	11/2/0
RAkEL	0.1912		7/6/0	10/3/0	10/3/0	8/5/0	10/2/1	10/3/0	10/3/0	11/2/0
CC	0.2315	0.2534		8/5/0	9/4/0	9/4/0	10/3/0	9/4/0	11/2/0	11/2/0
LP	0.0980 *	0.0374 **	0.1394		6/7/0	8/5/0	7/6/0	9/4/0	9/4/0	11/2/0
BR	0.0980 *	0.0180 **	0.1244	0.4034		7/6/0	7/6/0	9/4/0	8/5/0	11/2/0
HOMER	0.0980 *	0.2762	0.2315	0.3766	0.4861		7/6/0	9/4/0	10/3/0	10/3/0
CLR	0.0579 *	0.0346 **	0.1244	0.1912	0.4861	0.1727		7/6/0	7/6/0	10/3/0
BPMLL	0.0320 **	0.1727	0.1912	0.2108	0.2762	0.2762	0.4861		10/3/0	11/2/0
IBLR-ML+	0.0096 ***	0.0165 **	0.0232 **	0.0665 *	0.1107	0.0273 **	0.2315	0.0865 *		10/3/0
MLkNN	0.0023 ***	0.0023 ***	0.0029 ***	0.0029 ***	0.0044 ***	0.0044 ***	0.0096 ***	0.0026 ***	0.0196 **	

test for corresponding algorithm pair. The asterisks indicate the level of significance with which the null hypothesis can be rejected. The cells in the upper diagonal show the pairwise win/lost/tie counts for the corresponding pair. For example CascadeML has performed better than RAkEL and CC on 8 of the datasets and performed worse in 5 of the datasets.

CascadeML, when compared individually with other algorithms was found to be significantly better than MLkNN and IBLR-ML+ with the significance level of $\alpha = 0.01$, better than BPMLL with the significance level of $\alpha = 0.05$ and better than CLR, HOMER, BR and LP with $\alpha = 0.1$. Although in this isolated pairwise case, the null hypothesis could not be rejected on any significance level with RAkEL and CC.

It must be emphasised that this test, under this setting *cannot* be used to perform multiple classifier comparison without introducing Type I error (rejecting the null hypothesis when it cannot be rejected), as it does not control the Family Wise Error Rate (FWER) (García et al. 2010). Therefore, the p-values for each pair from this experiment should *only* be interpreted in isolation from any other algorithms.

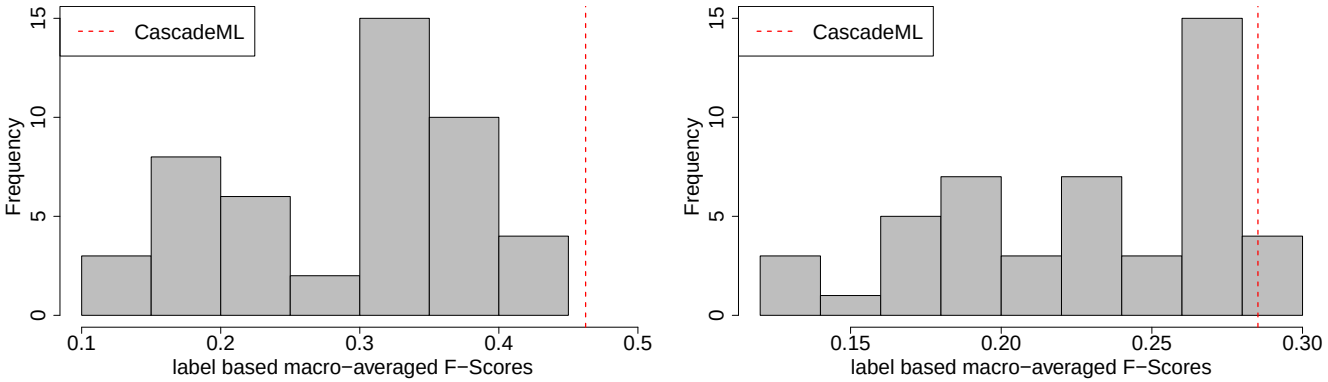
5.3 | CascadeML Architecture Analysis

It is important to note that CascadeML has the advantage of not requiring hyper-parameter tuning. For other algorithms the selection of hyper-parameter values can have a huge impact on performance. For example, Figure 4 shows the distribution of the label-based macro-averaged F-scores for different hyper-parameter combinations explored for RAkEL algorihm on *yeast* and *enron* datasets. The hyper-parameter combinations explored for RAkEL is label subset size and for the underlying SVM-RBFs are C and σ values. Note that the label based macro-averaged F-score values in Figure 4 vary significantly. The vertical red dashed lines in Figure 4 indicate the performance of the CasecadeML algoirithm on these datsets (only one value is shown as no hyper-parameter tuning is required). For the *yeast* dataset, CascadeML performed better than RAkEL for all hyper-parameter combinations considered. For *enron* dataset only 2.1% of the hyper-parameter combinations led to better results than CascadeML. In general the distribution skews towards models with relatively poorer performance. CascadeML attained similar high values of performance in both the cases (0.4624 for the *yeast* dataset and 0.2852 for the *enron* datset) without any hyper-parameter tuning, as the network learning algorithm discoveres the network architecture and uses adaptive learning rates while training.

The nature of the incremental growth and training in combination with the fast convergence nature of iRProp- with the L2 regularisation helped the network to generalise as well as converge faster. Also, note that the candidate unit pool size was set to 12 and all of them were run in parallel, hence the real runtime of the candidate training would be the maximum time taken among the 12 candidates in the candidate pool. Therefore, by exploiting the cascade architecture and training process, as well as using the iRProp- algorithm along with L2 regularisation, CascadeML was able to maintain a very good level of performance without hyper-parameter tuning.

Figure 5 shows the training costs of the Phase I cost function (as defined in Eq. (1)) for the *scene* dataset for one fold. The vertical dotted line indicates the addition of a candidate layer to the main network. After each addition of the candidate layer the cost increases but then sharply decreases before continuing to decrease more steadily.

CascadeML learns architectures with different numbers of nodes and different activation functions per layer. Different learned architectures trained on the same dataset can differ. For example Figure 6 shows a network learned by the CascadeML on the *yeast* dataset. For this specific execution, three cascaded layers were selected with L1 having 220 units and a tanh activation, L2 having 64 units and a linear activation, and L3 having 26 units and a linear activation. Some of the learned architectures may have a deeper network but fewer nodes per layer, whereas some learned architectures can be shallow but include more nodes per layer. The architectures of the networks learned using CasecadeML for the datasets used in the evaluation experiments are summarised in Table 5 and Figure 7 for every dataset over multiple folds. In Table 5 *Cascade depth* indicates the average depth of the cascade network (followed by standard deviation), and *Scaled hidden nodes* indicate the number of total hidden units divided



(a) yeast dataset **(b)** enron dataset

FIGURE 4 Histogram of label-based macro-averaged F-Scores achieved from all hyper-parameter combinations of subset size for RAkEL and C, σ for the underlying SVM-RBFs. The vertical dotted lines indicate the performance of CascadeML.

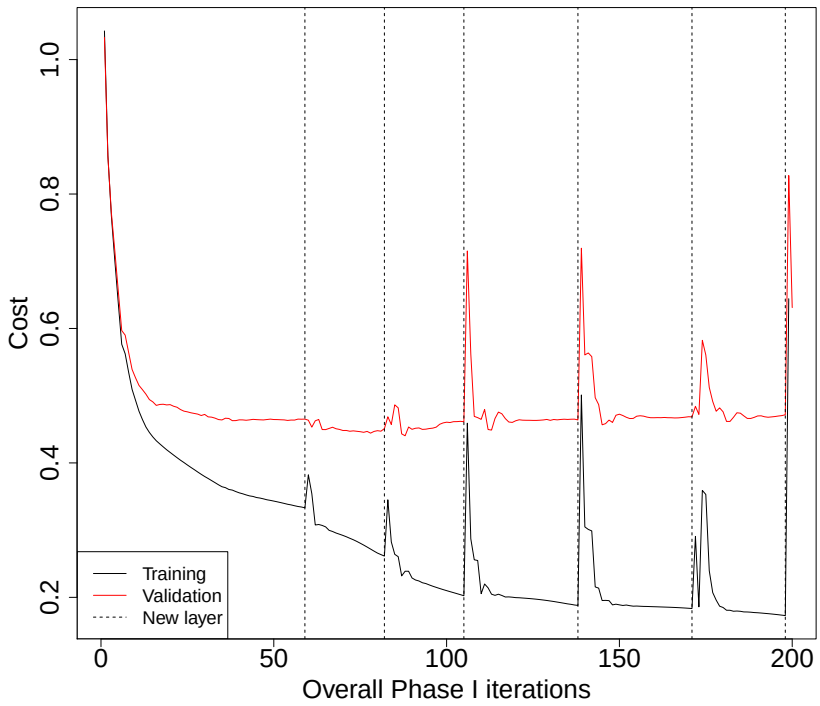


FIGURE 5 Training costs for a fold for scene

by the number of inputs for each dataset averaged over cross-validation folds (followed by standard deviation). Figure 7a shows boxplots of the learned network depths over folds for each dataset and Figure 7b shows the boxplots for the scaled hidden nodes.

It is interesting to note that for specific datasets, for each fold, the network architecture varies a lot, but the label based macro-averaged F-scores are consistent. This can be observed by noting that the standard deviations of the performances in Table 3 are small, the trained layer depth and the *scaled hidden nodes* values in Table 5 have high standard deviations. Figure 7c shows a scatterplot of the depths and the scaled hidden nodes values over all datasets and folds. This indicates that the learned networks were either deep with relatively fewer nodes per cascaded layer, or shallow but relatively more nodes per cascaded layer. This suggests that the networks had similar learning capacity and hence the label-based macro averaged F-score performance over the folds were similar, although the architectures learned were very different.

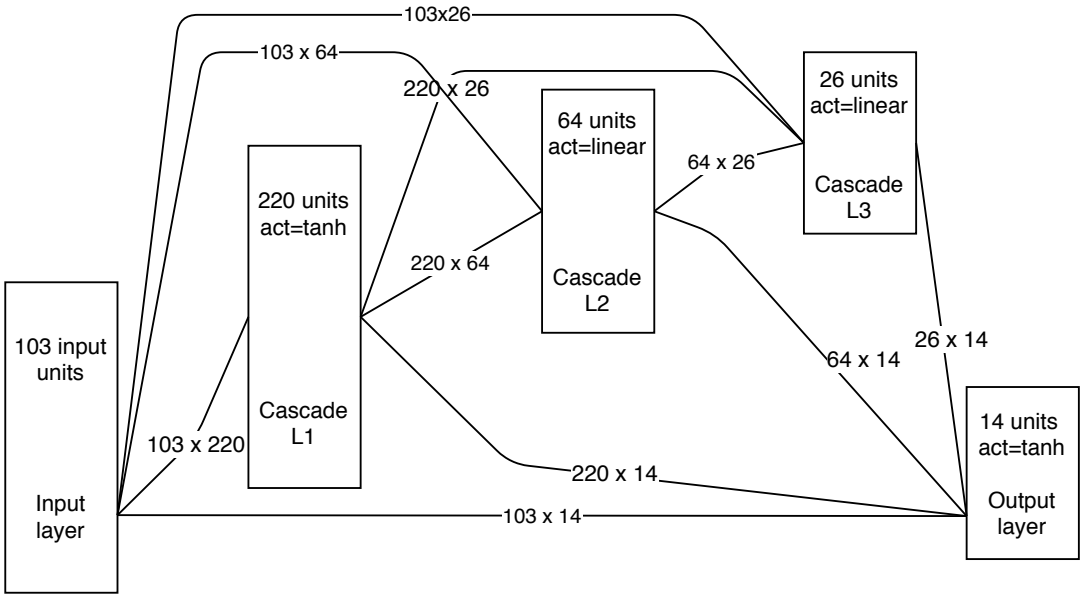


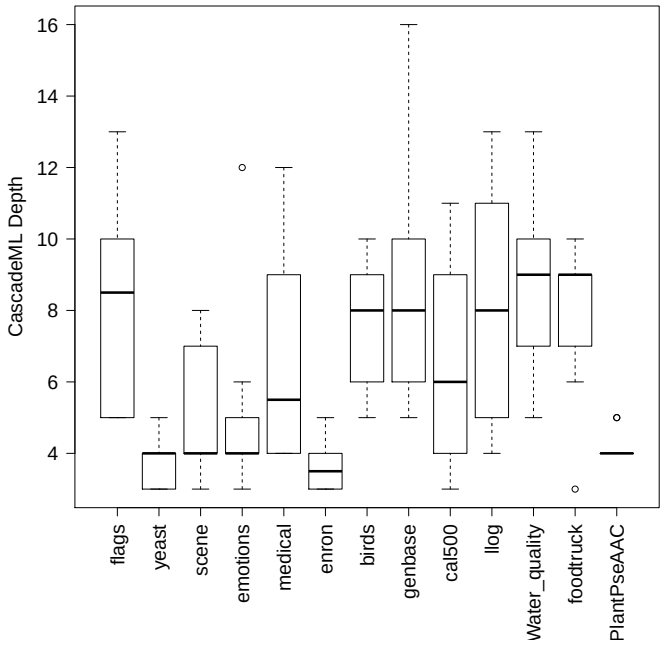
FIGURE 6 An example network generated by a run of CascadeML on the *yeast* dataset. The rectangles represent layers and the lines connecting the layers indicate full connection (the text on each line indicates the number of weights involved in that connection).

TABLE 5 Summary trained CascadeML network sizes for all datasets.

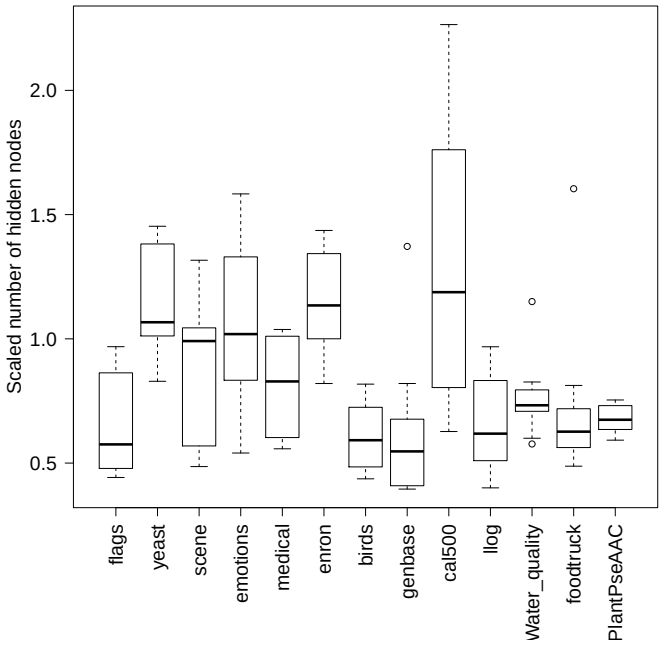
	Cascade Depth	Scaled Hidden Nodes
flags	8.3 ± 2.87	0.65 ± 0.21
yeast	3.8 ± 0.79	1.16 ± 0.22
scene	5.1 ± 1.85	0.89 ± 0.28
emotions	5.0 ± 2.58	1.04 ± 0.32
medical	6.8 ± 3.16	0.80 ± 0.20
enron	3.6 ± 0.70	1.16 ± 0.21
birds	7.6 ± 1.58	0.61 ± 0.13
genbase	8.4 ± 3.24	0.63 ± 0.30
cal500	6.4 ± 2.84	1.34 ± 0.60
llog	8.1 ± 3.45	0.67 ± 0.20
Water-quality	9.0 ± 2.36	0.76 ± 0.16
foodtruck	8.1 ± 2.18	0.72 ± 0.33
PlantPseAAC	4.2 ± 0.42	0.68 ± 0.05

6 | CONCLUSIONS AND FUTURE WORK

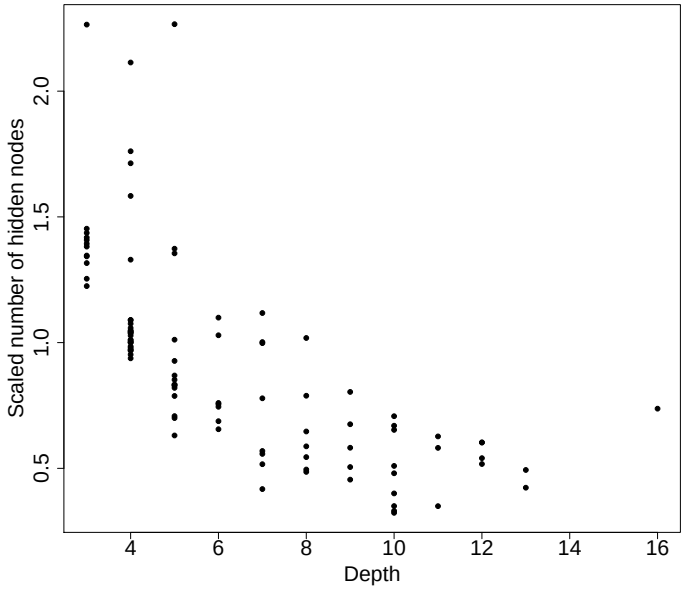
The work presented in this paper introduces a neural network algorithm, CascadeML, for multi-label classification based on the cascade architecture, which grows the architecture as it trains and takes inter-label associations into account. Except setting some bounds for the hyper-parameters,



(a) Cascaded depth



(b) Total hidden nodes scaled by input size



(c) All datasets, all folds depth vs. scaled number of hidden nodes

FIGURE 7 CascadeML trained network properties.

the method omits the requirement of hyper-parameter tuning as it automatically determines the architecture, and uses an adaptive first-order gradient descent algorithm, iRProp-. Due to the incremental nature of training, which trains different sets of the networks independently, and the use of an adaptive first-order gradient algorithm the training is very fast.

Evaluation experiments show that CascadeML attained the best average rank over thirteen multi-label datasets when compared with nine other multi-label classification algorithms. In this experiment all the other algorithms except CascadeML were hyper-parameter tuned. For CascadeML, as mentioned in Section 4.1, a few bounds were set. The experiment results show that CascadeML performed better on an average with respect to all the other algorithms based on average ranks. Although CascadeML had the best overall rank, the performance of RAkEL and classifier chains algorithms after being hyper-parameter tuned, were overall very close to CascadeML. When compared pairwise, CascacdeML performed significantly better than binary relevance, label powerset, BPMLL, MLkNN and IBLR-ML+.

CascadeML is the first attempt to develop an automatic neural network training algorithm for multi-label classification that requires minimal hyper-parameter tuning. Although CascacdeML's performance was good, there are several improvements which can be made to address in the cases where CascadeML did not perform well. A limitation of the BPMLL loss function used in CascadeML is that it cannot scale with increasing number labels (Nam, Kim, Loza Mencía, Gurevych, & Fürnkranz 2014). As the comparisons are pairwise, as the number of labels increase the computation becomes slow, as can be observed in the case of BPMLL. Therefore, it would be interesting to investigate alternative loss functions that can still take the inter-label associations into account without the need for expensive pairwise comparisons. It would also be interesting to study patterns in which layers grow during CascadeML training, so different mechanisms for adding new layers could be introduced to specifically focus on training separate sets of labels per cascade layer, instead of training them all at once.

ACKNOWLEDGEMENTS

This research was supported by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289_P2 and 15/CDA/3520

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in *MULAN* multi-label data repository at http://mulan.sourceforge.net/datasets-mlc.html and *Multi-Label Classification Dataset Repository* at http://www.uco.es/kdis/mllresources/

CONFLICT OF INTEREST

None

References

Baluja, S., & Fahlman, S. (1994, October). *Reducing network depth in the cascade-correlation learning architecture* (Tech. Rep. No. CMU-CS-94-209). Pittsburgh, PA: Carnegie Mellon University.

Boutell, M. R., Luo, J., Shen, X., & Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, *37*(9), 1757 - 1771.

Charte, F., Rivera, A., del Jesus, M. J., & Herrera, F. (2014). Concurrence among imbalanced labels and its influence on multilabel resampling algorithms. In *Hybrid artificial intelligence systems* (pp. 110–121). Cham: Springer International Publishing.

Chen, Z., Chi, Z., Fu, H., & Feng, D. (2013). Multi-instance multi-label image classification: A neural approach. *Neurocomputing*, *99*, 298 - 306.

Cheng, W., & Hüllermeier, E. (2009). Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, *76*(2-3), 211–225.

Cheng, W., & Hullermeier, E. (2009). Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, *76*(2-3), 211-225.

de Sá, A. G. C., Freitas, A. A., & Pappa, G. L. (2018). Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming. In *Ppsn*.

de Sá, A. G. C., Pappa, G. L., & Freitas, A. A. (2017). Towards a method for automatically selecting and configuring multi-label classification algorithms. In *Gecco*.

Fahlman, S. E. (1988). *An empirical study of learning speed in back-propagation networks* (Tech. Rep.).

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems 2* (pp. 524–532). Morgan-Kaufmann.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in neural information processing systems 28* (pp. 2962–2970).

Fürnkranz, J., Hüllermeier, E., Mencía, E. L., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine learning*, *73*(2), 133–153.

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, *180*(10), 2044–2064.

Gibaja, E., & Ventura, S. (2014). Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *4*(6), 411–444.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Grodzicki, R., Mańdziuk, J., & Wang, L. (2008). Improved multilabel classification with neural networks. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, & C. Poloni (Eds.), *Parallel problem solving from nature – ppsn x* (pp. 409–416). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hansen, L. K., & Pedersen, M. W. (1994). Controlled growth of cascade correlation nets. In M. Marinaro & P. G. Morasso (Eds.), *Icann '94* (pp. 797–800). London: Springer London.

Haykin, S. (1998). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Herrera, F., Charte, F., Rivera, A. J., & del Jesús, M. J. (2016). *Multilabel classification - problem analysis, metrics and techniques*. Springer.

Igel, C., & Hüsken, M. (2000). Improving the rprop learning algorithm. In *Proceedings of the second international icsc symposium on neural computation (nc 2000)* (Vol. 2000, pp. 115–121).

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. The MIT Press.

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, *45*(9), 3084 - 3104.

Nam, J., Kim, J., Loza Mencía, E., Gurevych, I., & Fürnkranz, J. (2014). Large-scale multi-label text classification — revisiting neural networks. In *Machine learning and knowledge discovery in databases* (pp. 437–452). Springer Berlin Heidelberg.

Nissen, S. (2007). Large scale reinforcement learning using q-sarsa (λ) and cascading neural networks. *Unpublished masters thesis, Department of Computer Science, University of Copenhagen, København, Denmark*.

Pakrashi, A., Greene, D., & Mac Namee, B. (2016). Benchmarking multi-label classification algorithms. In *24th irish conference on artificial intelligence and cognitive science (aics'16)*.

Phatak, D. S., & Koren, I. (1994, Nov). Connectivity and performance tradeoffs in the cascade correlation learning architecture. *IEEE Transactions on Neural Networks*, *5*(6), 930-935.

Prechelt, L. (1997). Investigation of the cascor family of learning algorithms. *Neural Networks*, *10*(5), 885 - 896.

Read, J., & Perez-Cruz, F. (2014). Deep learning for multi-label classification. *arXiv preprint arXiv:1502.05988*.

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

Rojas, R. (1996). *Neural networks: A systematic introduction*. Berlin, Heidelberg: Springer-Verlag.

Spyromitros, E., Tsoumakas, G., & Vlahavas, I. (2008). An empirical study of lazy multilabel classification algorithms. In *Proc. 5th hellenic conference on artificial intelligence (setn 2008)*.

Tsoumakas, G., & Katakis, I. (2007). Multi-label classification: An overview. *Int J Data Warehousing and Mining*, *2007*, 1–13.

Tsoumakas, G., Katakis, I., & Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ecml/pkdd 2008 workshop on mining multidimensional data (mmd'08)* (Vol. 21, pp. 53–59).

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., & Vlahavas, I. (2011). Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, *12*, 2411–2414.

Tsoumakas, G., & Vlahavas, I. P. (2007). Random k -labelsets: An ensemble method for multilabel classification. In *Ecml*.

Waugh, S., & Adams, A. (1994). Connection strategies in cascade-correlation. In *The fifth australian conference on neural networks* (pp. 1–4).

Wei, Y., Xia, W., Huang, J., Ni, B., Dong, J., Zhao, Y., & Yan, S. (2014). Cnn: Single-label to multi-label. *arXiv preprint arXiv:1406.5726*.

Wever, M., Mohr, F., & Hüllermeier, E. (2018). Automated multi-label classification based on ML-Plan. *CoRR*, *abs/1811.04060*.

Yu, Q., Wang, J., Zhang, S., Gong, Y., & Zhao, J. (2017). Combining local and global hypotheses in deep neural network for multi-label image classification. *Neurocomputing*, *235*, 38 - 45.

Zhang, M.-L., & Zhou, Z.-H. (2006, October). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Trans. on Knowl. and Data Eng.*, *18*(10), 1338–1351.

Zhang, M. L., & Zhou, Z. H. (2007). ML-kNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, *40*, 2038-2048.

Zhang, M.-L., & Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, *26*(8), 1819–1837.

Zhu, J., Liao, S., Lei, Z., & Li, S. Z. (2017). Multi-label convolutional neural network based pedestrian attribute classification. *Image and Vision Computing*, *58*, 224 - 229.

Zhuang, N., Yan, Y., Chen, S., Wang, H., & Shen, C. (2018). Multi-label learning based deep transfer neural network for facial attribute classification. *Pattern Recognition*, *80*, 225 - 240.

How to cite this article: A. Pakrashi, and B. Mac Namee (2020), A Multi-label Cascaded Neural Network Classification Algorithm for Automatic Training and Evolution of Deep Cascaded Architecture, *YYYY*, *XXXX*.