# Neural Architecture Search for extreme multi-label classification: an evolutionary approach

Loïc Pauletto, Massih-Reza Amini, Rohit Babbar, Nicolas Winckler

# Neural Architecture Search for extreme multi-label classification: an evolutionary approach

Loïc Pauletto
ATOS
University Grenoble Alpes
Grenoble, France
loic.pauletto@atos.net

Massih-Reza Amini
University Grenoble Alpes
Grenoble, France
Massih-Reza.Amini@univ-grenoble-alpes.fr

Rohit Babbar
Aalto University
Helsinki, Finland
rohit.babbar@aalto.fi

Nicolas Winckler
ATOS
Grenoble, France
nicolas.winckler@atos.net

## ABSTRACT

Extreme multi-label classification (XMC) and Neural Architecture Search (NAS) are research topics, which have gain a lot of interest recently. While the former deals in supervised learning problems with extremely large number of labels in text and NLP domain, the latter has been mainly applied to much smaller tasks, mainly in image processing. In this study, we extend the scope of NAS to (XMC) tasks. We propose a neuro-evolution approach, that has been found most suitable for a variety of tasks. The proposed NAS method automatically finds architectures that give competitive results to the state of the art (and superior to other methods) with faster convergence. Furthermore, the weights of the architecture blocks have been analyzed to give insight on the importance of the various operations that have been selected by the method.

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Information extraction.*

## KEYWORDS

Neural architecture search , Evolutionary algorithms , Natural Language Processing , Extreme multi-label classification

**ACM Reference Format:**
Loïc Pauletto, Massih-Reza Amini, Rohit Babbar, and Nicolas Winckler. 2020. Neural Architecture Search for extreme multi-label classification: an evolutionary approach. In *Proceedings of The Fourth International Workshop on Automation in Machine Learning (AutoML '20).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn


*AutoML '20, August 24, 2020, San Diego, CA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The main objective in XMC is to learn classifiers which can tag data with a set of relevant labels from an extremely large set of all possible labels. Machine learning problems consisting of hundreds of thousand labels are common in various domains such as tagging in large encyclopedias, product categorization for e-commerce [2], hash-tag suggestion in social media [4], and image-classification [8]. It has been demonstrated that, the framework of XMC can also be leveraged to effectively address ranking problems arising in bid-phrase suggestion in web-advertising and suggestion of relevant items for recommendation systems [6].

Various deep learning algorithms have been proposed in recent years for XMC using convolutional networks (XML-CNN in [10]), recurrent networks [11], and attention mechanism (AttentionXML in [16]). Even though these techniques have been relatively successful, the neural architecture design phase is complex and often requires human prior, with a good knowledge of the field and the data.

Over the last few years, NAS research has paved the way for the creation of dedicated neural architectures for a given task. In the literature, various methods and algorithms have been studied, among them some approaches use Reinforcement Learning (RL) as done by [17]. In order to speed up the search phase [13] have combined RL and weight sharing. We can also find among the other techniques some uses evolution mechanism [15] or based on the Bayesian optimization [7]. Large part of the studies, on NAS has focused on search algorithms for a small number of tasks (eg. image classification) and none of this have been applied on XMC before. In this study we propose `XMCNAS`, a NAS based method to automatically design architecture for the task of XMC, using a minimal prior knowledge. To evaluate our solution we use 3 large scale XMC datasets with an increasing number of labels. The discovered architecture gives competitive results with respect to the state of the art on the proxy dataset with a faster convergence. Then we transfer the best performing architecture to other datasets and evaluate it. We can summarize our contributions as following:

- We propose a domain specific *candidate operation* set for the NLP domain, with various operations which can act as different information extractor.

- In order to see which operation extracts information efficiently, we have examined the impact of operations on the final results by using a linear combination of trainable weights assigned at each operation output.
- The best derived architecture is more complex than the current state of the art one [16], but converge up to twice faster, with a similar training time. Moreover this architecture has shown competitive results with the state of the art

## 2 METHODS

In the following section we describe XMCNAS. Our approach is based on 3 main components: i) the embedding of the text, ii) the architecture search, and iii) the classification of the output. These three components form a pipeline in which components i) and iii) are fixed (i.e. be present in all networks) and are excluded from the search task. First, we present the search space and architecture search algorithm used. Then, we describe components i) and iii).

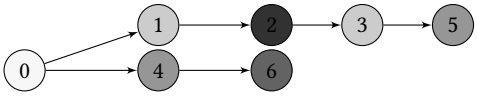



**Figure 1: Illustration of an architecture, with 6 layers. Node 0 represents the embedding layer. The numbers is the sampling order of the layers. The limit on the maximum number of previous layers that can be used as input is set to 5. In this example, therefore, layer 6 could only take as input nodes 1 to 5. We illustrate different operations with different colors.**

### 2.1 The search space

For the search space we use a *macro* architecture. The whole network can be represented as a Direct Acyclic Graph (DAG) of $K$ nodes, each node is an operation that are sequentially sampled. Each node can take only one input among the last 5 previous node. We initialize the set of previous with the embedding layer (i.e. the first 5 operations can take the embedding as input). For the *candidate operation* set, we have selected the most commonly used operations in NLP, which consist of a mix of convolutional, recurrent and pooling layers. We have defined four different versions of 1D-Convolutional layer, with a kernel size of 1, 3, 5 and 7 respectively. The convolution with $kernel = 1$ is comparable to a feed-forward layer. All convolution layers use a stride of 1 and use padding to maintain a coherent shape if necessary. We use two type of pooling layers which compute the average or the maximum over a filter size, the size of the filter 3 for both. As with convolution, pooling layers uses a stride of 1 and uses padding if necessary. We use as recurrent layers the two most commonly used versions, which are namely the Gated Recurrent Unit (GRU) [3] and the Long-Short Term Memory (LSTM) [5], which are able to capture long term dependencies. Specifically, we use bi-directional LSTM and GRU. Figure 1 illustrate an example of graph.

### 2.2 The search algorithm

As NAS algorithm, we use the *regularized evolution* as described in [15], which briefly consists in, the initialization of a trained population with architectures constructed randomly from sampled operations and then trained and evaluated. Sample randomly $N$ architectures, rank them by Precision@5 obtained on test set. The best performing one is selected for mutation, then the newly mutated architecture, is trained, evaluated and placed in the trained population. The oldest architecture is removed from the population. We chose this approach because it allows us to have a fine vision of the impact of each operation on the final result. Regarding mutation, we use the same as those defined by [15]:

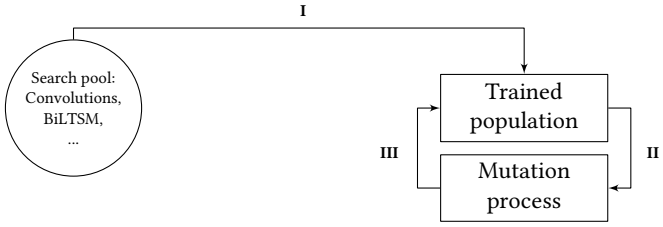



**Figure 2: I. Architectures are constructed from randomly sampled operations and then trained and evaluated, II. Sample randomly $N$ architectures, and rank them by Precision@5 obtained on test set. The best performing one is selected for mutation, III. The newly mutated architecture, is trained, evaluated and placed in the trained population. The oldest architecture is removed from the population.**

- Randomly choose an input of an node in the network and change it by a new input.
- Randomly choose an operation in the networks and change it by a newly sampled.

In order to see the impact of the number of layers on the final results, a third mutation, corresponding to the addition of a new layer, has been introduced. The choice among these mutations is random. Moreover, we combine these mutation with a linear combination of weights on the output of each layer. The weights of those combination are learned during the training process. The figure 2 is a visualization of the algorithm.

### 2.3 Embedding, Attention and classification modules

The network is made up of certain parts that are fixed namely the embedding, attention and classification modules.

*2.3.1 Embedding.* The embedding layer produces a fixed length representation, which means each words is transformed to a vector. We use the pre-trained embedding GloVe[1][12], with no fine-tuning during the training.

*2.3.2 Attention module.* Similarly as done in [16], we use a self-attention mechanism based on the one demonstrated in [9]. The purpose of the attention process helps to catch the important part of the text.

*2.3.3 Classification module.* The classification module is composed of 2 or 3 fully connected layers, and an output layer that classifies output into different labels.

## 3 FINDINGS

In this section we present the findings we got during this study. First, we present the datasets used. Then, we show the study we

[1] http://nlp.stanford.edu/data/wordvecs/glove.840B.300d.zip

**Table 1: Statistics of XMC datasets considered in our experiments. L: # of classes.**

| Dataset | # of Training examples | # of Test examples | L | Avg. of class labels per example | Avg. size of classes |
|---|---|---|---|---|---|
| EURLex-4K | 15,539 | 3,809 | 3,993 | 25.73 | 5.31 |
| Wiki10-31k | 14,146 | 6,616 | 30,938 | 8.52 | 18.64 |
| AmazonCat-13K | 1,186,239 | 306,782 | 13,330 | 448.57 | 5.04 |

conducted on the impact of the operations. Finally, we describe the XMCNAS discovered architecture, and the results we achieve with this architecture.

## 3.1 Datasets and evaluation metrics

We conducted our study on three of the most popular XMC benchmark datasets downloaded from the XMC repository[2]. These datasets are considered large scale, with the number of class labels varying from 4,000 to 30,000, which are listed from smallest to largest (in term of number of labels) by EURLex-4K, AmazonCat-13K, and Wiki10-31K summarized in Table 1. We followed the same preprocessing pipeline as used in [16]. As evaluation metrics we used the Precision at $k$ denoted by $P@k$, and the normalized discounted cumulative gain at $k$ denoted by $nDCG@k$ [6]. Both metrics are standard and widely used in the state of the art references.

## 3.2 Analysis of operation importance

During the search phase, we perform a weight analysis of the linear combination of output of each operation, and whether different operations combine effectively. Moreover, to extend this study we analyze the impact of i) the numbers of layers and ii) the operations when networks become deeper. Throughout the study, we scale the results of each layer with trainable weights of the linear combination.

**Combination of operations:** The purpose of this study is to determine which operation is the most important in the first layers. We have observed, in this analysis, that operation pairs of same kind, tend to have nearly equal weights (around 0.5 on each). However, some trends could be observed in the case of the combination of two convolutions, the one with larger kernel size have higher weights. This effect is particularly pronounced in the case of the kernel size of 1, which reflects the necessity of sequence modeling blocks at that level. In the case of mixed operations (BiLSTM and Convolution combined), it turned out that BiLSTM operations systematically have higher weights, on average 0.7 for the BiLSTM. More generally, our results show that architectures which contain BiLSTM at the first layer perform better ( $P@5 \in [0.59, 0.62]$). Compared to convolution based where the $P@5$ is a little less performing ($P@5 \in [0.56, 0.58]$). This indicates that the result is mainly based on the long-term dependencies captured by the BiLSTM rather than the local feature combinations generated by convolution.

**Study extension:** Concerning the impact of the number of layers, we calculated the average $P@5$ based on the number of layers. We varied the number of layers of from 2 to 6, and we observed that the averaged precision is nearly constant, regardless of the number of layers in the network, with results similar to those obtained in the previous paragraph. This result is corroborated by the previous analysis. We note in this configuration that the weights on additional layers are small compared to those that bypass it. For example, when we have 2 layers in the network in sequential order, the operations that take integration as input have a weight of 0.68, indicating that the most important information for the final results is extracted by the layer that take embedding as input.This trend has been observed in all experiments and suggests that, given our operations pool, additional layers does not bring much more information.

[2]http://manikvarma.org/downloads/XC/XMLRepository.html

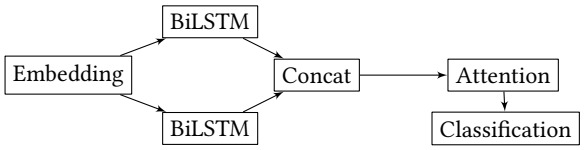



**Figure 3: The discovered network during our study, is composed of two BiLSTM which results are concatenated, then passed in an attention module and finally in a part of fully connected layers.**

## 3.3 Discovered network

The architecture found by XMCNAS, is made of two BiLSTM which take the embedding as input. The output of the two BiLSTM is then concatenated along the hidden dimension, and given as input of the attention module and finally to the classification module. The figure 3 is an visualization of the network.

## 3.4 Results

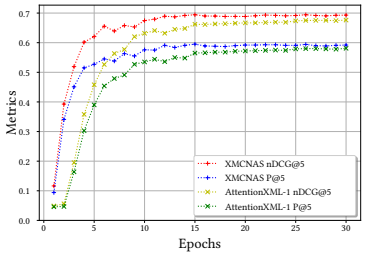
In this section we present the results obtained by the XMCNAS discovered network on various XMC datasets. First we present the results obtained on the EURLex-4K dataset used for the search phase, also named proxy dataset. Finally we evaluate the performance of this discovered architecture, transferred on the other datasets. To train our network we use 2 Nvidia GV100, with data parallelism training. We compare the results of our method to the most representative methods on XMC (with the implementation and the results provided by the authors). We compare our results with some state of the art methods XML-CNN [10], AttentionXML-1 [16], DiSMEC [1], and Parabel [14].

*3.4.1 EURLex-4K results.* On this dataset, the network got an improvement regarding the precision at $k$ to the state of the art. As presented in the left hand side of the Table 2 we got an improvement on P@1, P@3 and P@5. A significant improvement is obtained on precision at 3 and 5, where we get 0.738 and 0.620 respectively compared to 0.730 and 0.611 before. Figure 4a presents the evolution of P@5 and the nDCG@5 on the validation set with respect to the number of epochs. We observe on figure 4a that our network have a faster convergence. The results is obtained around 15 epochs and after this point, we only get small improvement which indicates that the network might overfit.

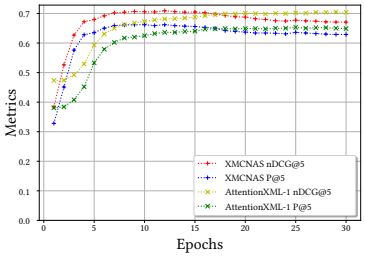
**Table 2: Comparison performance table on datasets. Best results are shown in bold.**

| | EURLex-4K | | | Wiki10-31K | | | AmazonCat-13K | | |
|---|---|---|---|---|---|---|---|---|---|
| Methods | P@1 | P@3 | P@5 | P@1 | P@3 | P@5 | P@1 | P@3 | P@5 |
| DiSMEC | 0.832 | 0.703 | 0.587 | 0.841 | 0.747 | 0.659 | 0.938 | 0.791 | 0.640 |
| Parabel | 0.821 | 0.689 | 0.579 | 0.841 | 0.724 | 0.633 | 0.930 | 0.791 | 0.645 |
| XML-CNN | 0.753 | 0.601 | 0.492 | 0.814 | 0.662 | 0.561 | 0.932 | 0.770 | 0.614 |
| AttentionXML-1 | 0.854 | 0.730 | 0.611 | **0.870** | **0.777** | 0.678 | **0.956** | **0.819** | **0.669** |
| **XMCNAS** | **0.858** | **0.738** | **0.620** | 0.849 | 0.772 | **0.681** | 0.951 | 0.813 | 0.664 |

*3.4.2 Transferred architecture results.* We train and evaluate the discovered architecture following the same training procedure as for EURLex-4K, on the two other datasets. The middle and right hand side of table 2 show the comparison of the architecture found by XMCNAS on EURLex-4K with others methods. We notice that the discovered architecture transferred to larger datasets get close results to the current state of the art. In some case we slightly surpass the results like in the $P@5$ on the Wiki10-31K. Moreover we notice in Fig. 4b the same trend as the proxy data set, that our method has a faster convergence, this was also the case for AmazonCat-13K. Ours results presented in table 2 on Wiki10-31K and AmazonCat-13K are obtained in half of the epochs required by AttentionXML-1.




**(a) EURLex-4K**




**(b) Wiki10-31K**

**Figure 4: Plot of the nDCG@5 and P@5 on the validation set, on two different datasets. We notice, the discovered architecture have a faster convergence compared to AttentionXML-1 [16]. In the 4a our method get better final results, in 4b our final results (around epoch 15) are close.**

## 4 SUMMARY AND FUTURE WORK

We have presented in this work an automated method to discover architecture for the task of XMC, based on the regularized evolution [15] and with a domain-oriented pool of operations. XMCNAS has found architecture that provided competitive results with the existing state of the art methods [16], and in some cases overpassed them. Moreover, the discovered network showed faster convergence rates on all datasets, despite being more complex than the actual SoTA. Possible future steps include the development of a method that can handle the search phase on large scale datasets, the extension of the search space by adding attention mechanism operation, the tuning of hyper-parameters, as well as methods to speed up the search process.

## REFERENCES

[1] Rohit Babbar and Bernhard Schölkopf. 2017. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 721–729.
[2] Samy Bengio, Jason Weston, and David Grangier. 2010. Label Embedding Trees for Large Multi-Class Tasks. In *Neural Information Processing Systems*. 163–171.
[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
[4] Emily Denton, Jason Weston, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. 2015. User Conditional Hashtag Prediction for Images. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[6] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD ICKDD*. 935–944.
[7] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD ICKDD*. 1946–1956.
[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*. 1097–1105.
[9] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).
[10] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR*. 115–124.
[11] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. 2017. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In *Advances in neural information processing systems*. 5413–5423.
[12] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
[13] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).
[14] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*. 993–1002.
[15] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.
[16] Ronghui You, Suyang Dai, Zihan Zhang, Hiroshi Mamitsuka, and Shanfeng Zhu. 2018. Attentionxml: Extreme multi-label text classification with multi-label attention based recurrent neural networks. *arXiv preprint arXiv:1811.01727* (2018).
[17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.