HA-Growing Tree-CNN for Hierarchical Image Recognition. Real Application on Whole Histological Images.

Pietro Barbiero, Chiara Sopegno, Antonio Tavera

Abstract

Keywords: cancer, CNN, deep learning, hierarchical models, incremental learning, growing models, neural network, transfer learning, tree-CNN, WSI

1. Introduction

1.1. Biological Background and Whole Histological Images

Historically, histology involved the study of human cells and tissues using microscopy. The visual analysis of such samples allowed pathologist to look for abnormalities and clinicians to decide for a diagnosis. In recent years, the spread of Whole Histological Images (WSIs) has allowed the use of more powerful IT tools. WSIs are digitalized multi-resolution images of human tissues obtained via surgical operation. The advantages of WSIs and IT methods include automation and time saving processes together with (in some cases) human level performance.

1.2. IT Background and Objective

One of the most powerful IT tools available for image processing and recognition are neural networks. In particular, deep Convolutional Neural Network (CNN) emerged as leading technology for image classification. Such deep structure is able to automatically extract features from large image data sets and use them in order to correctly classify new images. Some of these

Email addresses: pietro.barbiero@studenti.polito.it (Pietro Barbiero), chiara.sopegno@studenti.polito.it (Chiara Sopegno), antonio.tavera@studenti.polito.it (Antonio Tavera)

CNNs (such as AlexNet [1] and GoogleNet [2]) are able to recognize tens or thousands of different images, such as plants, cars, animals, etc. However, CNNs are *flat models*, i.e. the external user cannot easily understand how the model groups batches of classes and explore data at different granularity levels. Moreover, such neural networks are not able to infer human-like rules such as 'both women and men are human beings'. Besides, in some cases, the user may need to incorporate his own knowledge to improve recognition when data are incomplete or to improve the model intelligibility. Last but not least, the training process of deep CNNs often takes a long time and erases the network memory, discouraging the insertion of new data at short time intervals. Some of these drawbacks were addressed by the Tree-CNN algorithm. In [3], authors proposed a hierarchical classifier composed of CNNs. At higher tree levels, CNNs try to recognize more abstract classes (such as animals and cars) while at lower levels, they try to perform a finer classification (such as between women and men). The objective of this work is to improve the Tree-CNN approach and this will be done by the HA-Growing algorithm (Human Adaptive-Growing), that enhances the growing process and integrates the data-driven approach with human knowledge, allowing user interaction during the training process.

2. State of the Art of Hierarchical Neural Networks for Image Recognition and Incremental Learning

2.1. Convolutional Neural Networks

A Convolutional Neural Network (a.k.a. CNN or ConvNet) is an artificial neural network specialized in image analysis. Its basic structure is composed of a sequence of different kind of layers [4]:

- **Convolutional layer**. It is the characteristic layer of this neural network. It takes as input a digital image, represented as a matrix of numbers (pixels). The input is convolved with a smaller (usually squared) matrix of weights (a.k.a. filter). The operation consists in sliding the weight matrix along the input image. For every position, the element-wise multiplication between the weight matrix and the corresponding part of the input image is computed. The values of the resulting matrix are added together, generating a single number. These output numbers are inserted in order in the output matrix (a.k.a. convolved feature or feature map). More intuitively, the filters act as feature extractors on

the input images, detecting edges, shapes, lines and textures, according to their weights.

- **Activation layer**. This layer introduce non-linearity inside the neural network model, applying non-linear function element-wise on the feature map. The most used ones are ReLU, tanh and sigmoid functions.

- **Pooling layer** (a.k.a. subsampling). This layer reduces the size of the feature map, retaining only the most important information. It takes the largest element (Max Pooling), the average of the elements (Avg Pooling) or the sum of the elements (Sum Pooling) in a spatial neighborhood of the feature map.

- **Dense layer** (a.k.a. fully connected). It is a standard feed-forward neural network layer were every neuron in the previous layer is connected to every neuron in the next one.

- **Output layer**. It is the last layer of a CNN architecture. It usually applies the SoftMax function on the previous layer outputs in order to perform multi-class recognition.

Convolutional and dense layers contain the model parameters. It is not unusual to see models with millions of parameters. The training process of the neural network consists in optimizing these parameters in order to minimize the misclassification error. Modern optimization algorithms, such as Adam [5] and Adagrad [6], are based on stochastic versions of the gradient descent (SGD) algorithm to speed-up the training process. Overall, the steps of a CNN algorithm are:

- **Initialization**. The network parameters are initialized using random (and usually small) values.

- **Forward propagation**. An input sample (one image) is shown to the network, whose output layer provides the probabilities of class membership using the SoftMax function.

- **Loss Evaluation**. The probabilities of class membership ($\hat{y}^{(i)}$) are compared with the true label of the image ($y^{(i)}$) to compute the loss function, like the squared error function:

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = (y^{(i)} - \hat{y}^{(i)})^2 \tag{1}$$

- **Backpropagation of the error**. Starting from the last layer to the first one, the network parameters are updated through a gradient descent-like algorithm using the error gradient with respect to the layer parameters.

- **Repetition**. The process (forward propagation, loss evaluation and backpropagation) is repeated for each image. Each data set presentation to the network is called *epoch*. The training process usually involves more than 1 epoch to have a good optimization of the weights.

2.2. Tree-CNN Approach

Tree-CNN is a classification model inspired both from hierarchical classifiers and from Convolutional Neural Networks. The model architecture is a tree where each internal node is a classifier. Internal nodes placed higher along the hierarchy distinguish among more abstract classes, while finer classifications are demanded to deeper nodes. Each leaf node is instead uniquely associated to a class label. When the classification of a new image is required, it is fed into the tree and sequentially classified until it reaches a leaf node and the corresponding predicted label. It's important to note that all the path followed by the image from the root to the leaf node provides useful information for the hierarchical classification. When instead data belonging to new classes are acquired, it is necessary to expand the tree architecture and this can be done through the procedure described in [3].
A small sample ($\approx 10\%$) is randomly selected from the training set of the M new classes. At each node (except for leaves), these new images are presented to the CNN of the node, one class at a time. For each new image, the output of the CNN is a k-dimensional vector, where k is the number of old classes (corresponding to the number of children of the current node). Each element of this vector denotes the probability that the new image is classified as class k by the CNN. However, in the implementation of the algorithm for the expansion of the tree, the output of the CNN that is considered is the one $before$ the Softmax function is applied. Having fed all the new images one class at a time, the algorithm build the tensor $O^{K\times M\times I}$, where I is the number of new images per class. Therefore, $O(k, m, i)$ is a score representing how likely is that the sample i of class m (new class) is classified as class k (old class) by the CNN. Taking the average along the dimension I, authors obtained the $2d$-matrix $O_{avg}^{K\times M}$. Then they applied the SoftMax function to this matrix to obtain the likelihood matrix $L^{K\times M}$, which indicates how

strongly the new class m associates on average to the old class k. Besides, they rearrange the columns l_m of $L^{K \times M}$ such that:

$$S = [l_1, l_2, \ldots, l_M] \tag{2}$$

$$\max_k l_1(k) \geq \max_k l_2(k) \geq \cdots \geq \max_k l_M(k) \tag{3}$$

By analyzing the matrix S one column at a time, they append the new class to the root node according to three rules:

- **add the new class to an existing child node** if $\max l_m$ is greater than a user defined threshold t_u
- **combine one or more child nodes and the new class to form a new child node** if more than one element of l_m exceeds the threshold t_u, i.e. the new class is similar to more than one of the old classes
- **add the new class as a new child node** if $\max l_m$ is lower than t_u

The process is repeated until leaf nodes are reached.
The Tree-CNN approach combines the power of deep CNN together with the characteristic interpretability of decision tree classifiers. The hierarchical structure allows users to understand at a glance if some classes are similar and can be grouped together. Besides, the incremental learning behavior simplifies the use of the system in real applications, where data are continuously generated and need to be processed shortly.
However, in our opinion, the Tree-CNN algorithm shows some weak points that we try to address in the rest of the report. In particular, we will focus on:

- **additional ways of growing**. The growing rules of Tree-CNN try to preserve most of the original tree structure. We think that in some cases the best choice is to increase the computational time in exchange for more accurate results.
- **human interaction**. In some cases, the information contained in the input data is not sufficient to make the best growing choices. We think that domain knowledge and human intuitions could help AI. Therefore, we allow external user interaction during the growing process when AI hesitates.

3. HA-Growing Tree-CNN

3.1. Dataset

Our algorithm will analyze datasets that typically assume a hierarchical structure, i.e. they contain classes that are in a subclass-superclass relation between each other. These datasets are usually organized in subfolders, each one containing data on a different level of the hierarchy (for example, a first folder can contain data about the two classes "Vehicle" and "Animal" and a second one data about the classes "Truck", "Car" and "Frog"). The division in folders may be due to two different factors:

- previous knowledge on the hierarchy, that allowed to separate classes before starting the incremental learning procedure;
- new data arriving over time in a real application, for example after new experiments or technical improvements.

The subfolders will be fed into the model at different stages and each insertion will cause the tree structure to grow.
We will focus in particular on colorectal cancer data, but the analysis can be generalized to any kind of hierarchical database. Our dataset consists of five classes of interest that we want to classify as best as possible: Adenocarcinoma (AC), Healthy (H), Serrated (S), Tubular (T) and Villous (V). The last three categories are subclasses of Adenoma (AD), which is a precursory lesion that is still not cancerous but can turn into cancer in following stages. The organization of our dataset however will be the following:

- H-AC data, which divide images of cancerous tissues from not cancerous ones;
- H-AD data, which distinguish the images of Adenoma from the ones of actually healthy tissues;
- S-T-V data, which split data relative to Adenoma in the three subclasses.

This division has been made thanks to the former human knowledge, that allowed us to distinguish different levels of the hierarchy. We can see that the class H appears in two different subfolders, however that label doesn't correspond to the exact same class in the two cases, as in H-AC it denotes a

'not cancerous tissues', while in H-AD it denotes 'actually healthy tissues'. For this reason we decided as a first preprocessing step to rename the class H in H-AD as HH, which will therefore be a subclass of the class H in H-AC and will denote actually healthy tissues.

3.2. Network Architecture

The HA-Growing Tree-CNN is a hierarchical classifier composed of deep CNN that grows dynamically according both to data and user needs. The base classifier is Residual CNN with 50 layers (a.k.a. ResNet50). Despite classical CNN, Residual CNN directly connects input of n^{th} layer to some $(n+x)^{th}$ layer. It has been proved that training this form of networks is easier than training simple deep convolutional neural networks. We downloaded the ResNet50 weights through the Keras framework and discarded the fully connected part of the network. We froze the learning for the first 45 ResNet50 layers, which already learned basic features of images, and we added a new fully connected network on top composed of a single layer with 1024 neurons. For the activation function of the dense layer we used the ReLU function. We also added dropout layers with rate 0.8. As loss function we used the cross-entropy error function and as optimizer the Adam algorithm, which proved to perform better on our data set with respect to other SGD algorithms like Adagrad or RMSprop. In this way we implemented a transfer learning and a fine tuning of the ResNet50 using the WHI data set.
Hyperparameters has been set through various experiments that allowed us to choose the ones that best performed on the type of data that we had: this includes parameters of the net like dropout values and number of layers to be fine-tuned, as well as training parameters like batch size, learning rate and number of epochs. For the training of the ResNet50 we chose to use a learning rate of 10^{-4}, a batch size of 100 and a maximum number of epochs of 40. For what concerns the epochs, we decided to implement an early stopping criterion that stops the training when the validation accuracy doesn't improve for more than 5 epochs and that allows us to avoid overfitting our data.

3.3. Data Structure

The data structure we will use for describing the tree-CNN is basically the same presented in section *2.2* and described by authors in [3]. Each internal node of the tree contains a CNN that performs a classification once an image is fed into the node. The output of the net represent the probabilities that the image belongs to each of the children nodes. The image is then fed into

the node with the higher probability. If it is again an internal node the procedure is repeated again. Instead, if it is a leaf node (which does not contain any CNN) the label of that node is assigned to the image. However we will deal with two types of internal nodes: all of them will contain a CNN and perform classification, but some will also be characterized by a label denoting a superclass, while others will be just branch nodes that don't add any further information about the image. These branch nodes will be a representation of all their children and may be thought as abstract classes. We will see in section *3.4* how these two types of nodes may be generated.

3.4. Growing algorithm

Our algorithm is designed to sequentially take data associated with classes still not known from our classifier and incrementally expand the tree structure. The first step is always to train the root node to classify between two or more classes given as first input. When data belonging to different new classes have been acquired, the network need to learn to classify them. This can be done in two different way:

- by inserting all the new classes together (*batch insertion*)
- by inserting each of the new classes separately so that each one can be inserted in different positions along the tree (*single class insertion*)

As we have seen in section *2.2*, when we want to add new classes to the tree the first thing is to take a small sample of new data to compute the likelihood matrix $L^{K\times M}$, where K is the number of old classes classified by the root node (or by another internal node if we are in another step of the tree-growing algorithm), and M is the number of the new classes being inserted. Each element l_{km} indicates how strongly the new class m associates on average to the old class k. The kind of exploitation of this matrix depends on the type of insertion the user chooses. Additional procedures of computing the likelihood matrix will be discussed in section *3.4.4*.

3.4.1. Batch insertion

This option can be chosen by the user if he knows the new classes inserted have a strong relationship and should remain grouped together. In this case, once the likelihood matrix L is computed, we only look at the maximum element $\mu = \max_{k,m} l_{km}$ and we compare it with a threshold $t_u \in (0, 1)$ defined by the user. We have different options for the batch insertion:

- if $\mu > t_u$, the classes are added to the child $\tilde{k} = \text{argmax}_k\, l_m(k)$ of the current node. Adding the classes to the child $\tilde{k}$ means that if it is still an internal node the procedure will be repeated as before, if it is instead a leaf node, it will be turned into an internal node, with a CNN able to distinguish between the M new classes, and all the new classes will be added as children of this node.

- if $\mu < t_u$ instead, the new classes will not be added to any of the children of the current node since they are not enough similar to any of them. So we have now two possibilities:

  – first we want to evaluate if these new classes can be seen as superclasses of the ones that they are being compared to. This is checked first by creating a new separate root node with a CNN that distinguish between the new classes. Then we will apply the same procedure used before, but switching the roles of new and old classes. If the obtained likelihood matrix has now an element that overcome the threshold, then the old classes will be inserted as children of the corresponding node and the new root node will be inserted in the tree in the position in which previously was positioned the father node of the old classes.

  – if instead also this second attempt of insertion is not successful, the new classes will be inserted at the same level of the old ones. All the classes will be children of the same node and the CNN of this node will be retrained to distinguish between all the classes.

The batch insertion may seem more restrictive than the single class one, however this implementation actually allows to overcome the rigid structure of the tree that would not allow new superclasses to be added in the middle of the tree over already existing nodes.

3.4.2. Single class insertion

The option of inserting one class at a time like described in [3] gives more freedom in order to fit each class in the tree as best as possible, especially if new classes have big differences among each other. In our algorithm we maintain the basic structure described by [3] authors. Given a column l_m of the likelihood matrix and a user defined threshold t_u, we have three possibilities:

- adding the new class to an existing child node, if $\max l_m$ is greater than the threshold t_u

- combining one or more child nodes and the new class to form a new child node, if more than one element of l_m exceeds the threshold t_u
- adding the new class as a new child node, if $\max l_m$ is lower than t_u

However, we need to define what "add a new class to an existing node" exactly means. If the node is a branch node with no label on it, we just repeat the same procedure as before. If the node has instead a label on it (the node could be both a leaf node or an internal node associated to a class), the new class is inserted below only if it turns out to be a subclass of the class of the node (the superclass-subclass relation can be specified by the user in an interactive way). If instead the new class and the class of the current node are not related to each other, a new branch node is created with a CNN that will distinguish between the two classes. This branch node will take the place of previous labelled node, while the labelled node with its subtree will be inserted as child of this new branch node together with new class.

3.4.3. AI-Human Interaction

As previously discussed, along the algorithm is possible to integrate the automatic growing of the tree with human knowledge. The first step in which the user can interact is choosing between batch and single class insertion, so that he can decide if certain data should remain together or not. We have also a second important interaction of the user when he specifies whether or not a certain class is a subclass of another. This aspect is really important in order to maintain all the classes of interest on the leaves of the tree. If for example the algorithm classifies the class AD as very similar to AC, if we insert the class AD as subclass of AC, we will have a classification of the images which is incoherent with the knowledge in the pathological field. We could have for example an image classified as AD, that had been in the first place classified as AC, and that can lead to dangerous errors in classifications. We also provided the user with the opportunity of specifying all the already known relationship before running the algorithm, in order to make the interaction more efficient.

3.4.4. Alternative likelihood matrix

We have tried an alternative way of computing the values of the likelihood matrix $L^{K\times M}$. For this purpose we have decided to implement a statistical procedure of mean comparison between two distributions. We will make use

specifically of the two statistical tests known as Welch's t-test and MannWhitney U-test.
Each element l_{km} of L is computed by taking the same number of samples from the training sets of the two classes k and m that we are considering, feeding them into the current CNN and applying one of the two statistical test on the scores (before the softmax) returned by the net. Both tests perform a comparisons and provide a statistical measure, called p-value, of how much the two distributions of scores are significantly different one from the other. Hovever, the Welch's t-test requires the two distributions to be normally distributed: it is in fact an adaptation of the more famous Student's t-test to the case of unequal variances between data samples. The Mann-Whitney U-test is instead a non parametric test that will be used in case of not normally distributed values. Therefore the first test that has to be made is a normality check of all the distributions of data used to compute L and the next procedure to be used is decided accordingly.
In a t-test the p-value measures if the difference in mean between the two distributions is statistically significant. In an accurate statistical test we would use a threshold under which we would refuse the so called *null hypothesis* that the two population means are equal. If the p-value is under the threshold we can affirm that the two means are different, while theoretically nothing can be told if the p-value is above that threshold. In our implementation we decided to take an approximation of this concept and just use the p-value as a measure of similarity between the two distribution: a higher p-value means a higher similarity between the mean of the two distributions and viceversa. When we feed the samples relative to the classes k and m into the CNN we obtain two sets of scores. Performing the t-test on these two sets of scores, we obtain a p-value measuring how similar they result in relation to the CNN model in the current node. This p-value is inserted as element l_{km} of the matrix L and then compared with prefixed thresholds as described above.

3.5. Further improvements

In addition to this basic algorithm we decided to exploit two more procedures in order to enhance the performance of our algorithm:

- *analysis of variance* on the scores given by a CNN, in order to choose in each case the best option between batch and single class insertion;
- some basic *data augmentation* techniques.

3.5.1. Analysis of variance

Since not always human knowledge allows to discern if it is better to perform a batch insertion or instead separate the classes, we tried also to perform a one-way analysis of variance on the scores obtained when feeding the data of new classes in an already existing CNN. One-way ANOVA allows to detect if the variability between samples of new data is already explained by our current model (the CNN of the current node) or not. We take evenly sized data samples from each new class, we feed them into the current CNN and we perform the test on the sets of scores that we obtain. If all the variability of the new classes was explained by the model, then their sets of scores would have the same distribution.
The ANOVA tests the null hypothesis that the scores of the samples have distributions with same mean values and this hypothesis is rejected if the p-value is lower than a fixed threshold (ex. 10^{-3}). If we reject the null hypothesis, then we proceed with single class insertion, since the score samples result to be significantly different one from the other. If we do not reject the null hypothesis, it means that the current model seems to already explain the internal variability between the samples of the new classes and so we can proceed with batch insertion.
However a one-way ANOVA can be performed only under the assumptions that the samples of scores are independent and that they come from normal distributions with equal variance. If these assumptions are not met, we can use instead the Kruskal-Wallis H-test, even if it implies some loss in precision.

3.5.2. Data augmentation

We have also observed that performing data augmentation could be helpful in our framework for two different reasons:

- to improve the accuracy of the CNNs in case the quantity of data available was not enough big to allow a good training;
- in order to balance classes at each level of the tree, since for some of them the amount of data available is less than for some others.

For this reason we have tested the `imageDataGenerator` class provided by the Keras API in order to perform different combinations of:

1. feature-wise centering and standardization;
2. random rotations, shifts, flips and shear;
3. random zoom and channel shifts.

4. Results and Discussion

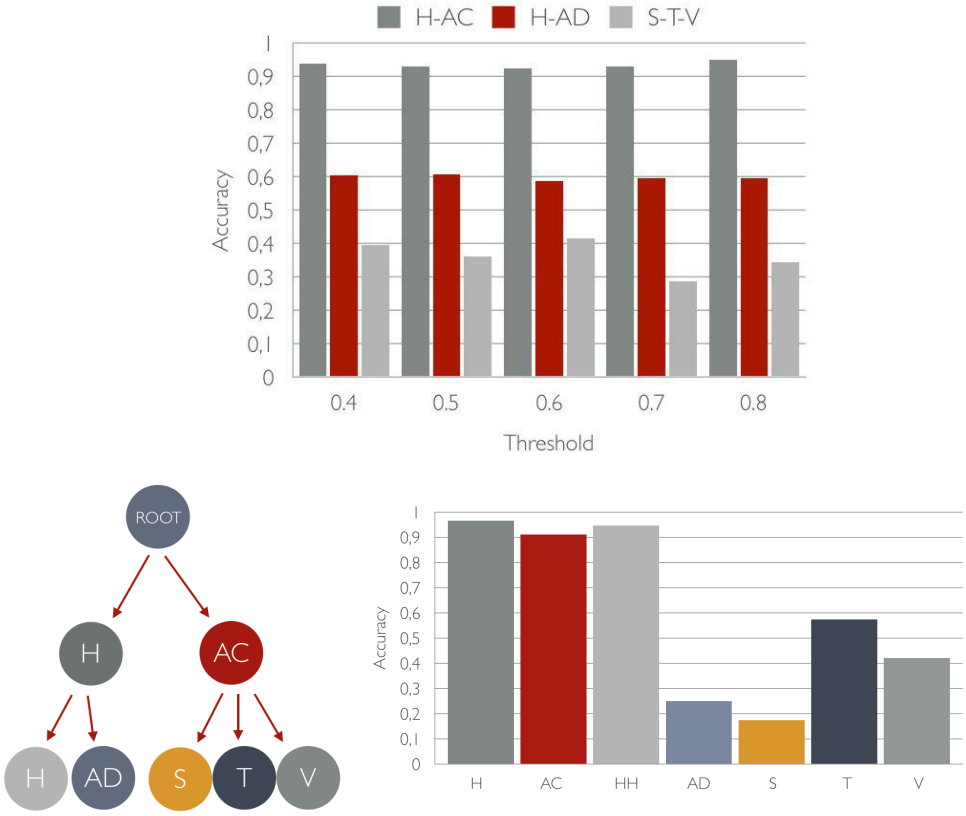

Figure 1: Outcomes using the batch insertion method. The figure on the top shows the classification accuracies for several choices of t_u. The figure on the bottom-left shows the tree structure corresponding to $t_u = 0.6$ (best value on average), while the one on the bottom-right shows the related accuracies.

In this section we present the results of our experiments. The final neural tree performance is evaluated on unseen test sets. Besides, given the peculiarity of the hierarchical classifier, we decided to evaluate as correctly classified each sample whose path down the tree passes through the node corresponding to its class.
In the first experimental setting we grouped training samples according to their class labels in three batches:

- H-AC

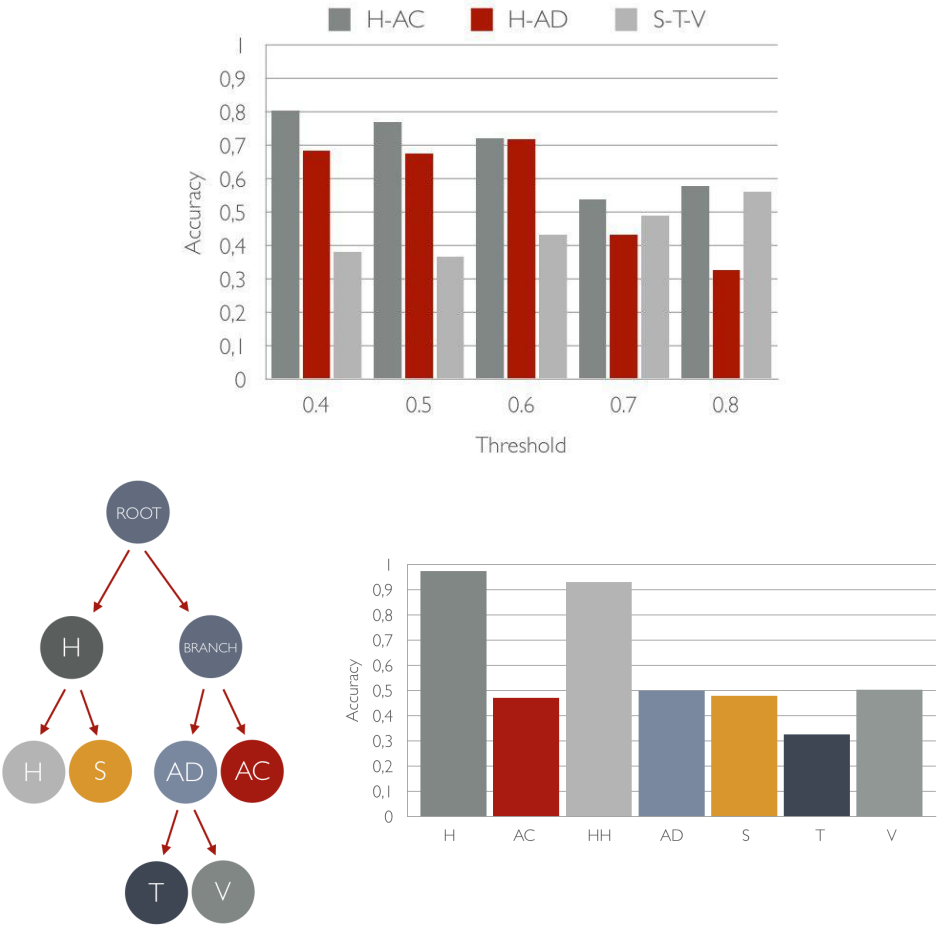

Figure 2: Outcomes using the single class insertion method. The figure on the top shows the classification accuracies for several choices of t_u. The figure on the bottom-left shows the tree structure corresponding to $t_u = 0.6$ (best value on average), while the one on the bottom-right shows the related accuracies.

- H-AD
- S-T-V

Each group is then presented to the network, following the above order. This experimental setting is then repeated for several choices of the user defined threshold. Fig. 1 shows a summary of the outcomes. Interestingly, the resulting tree structure is the same regardless of t_u choices. The training process always generates a hierarchical structure where H-AD classes are ap-

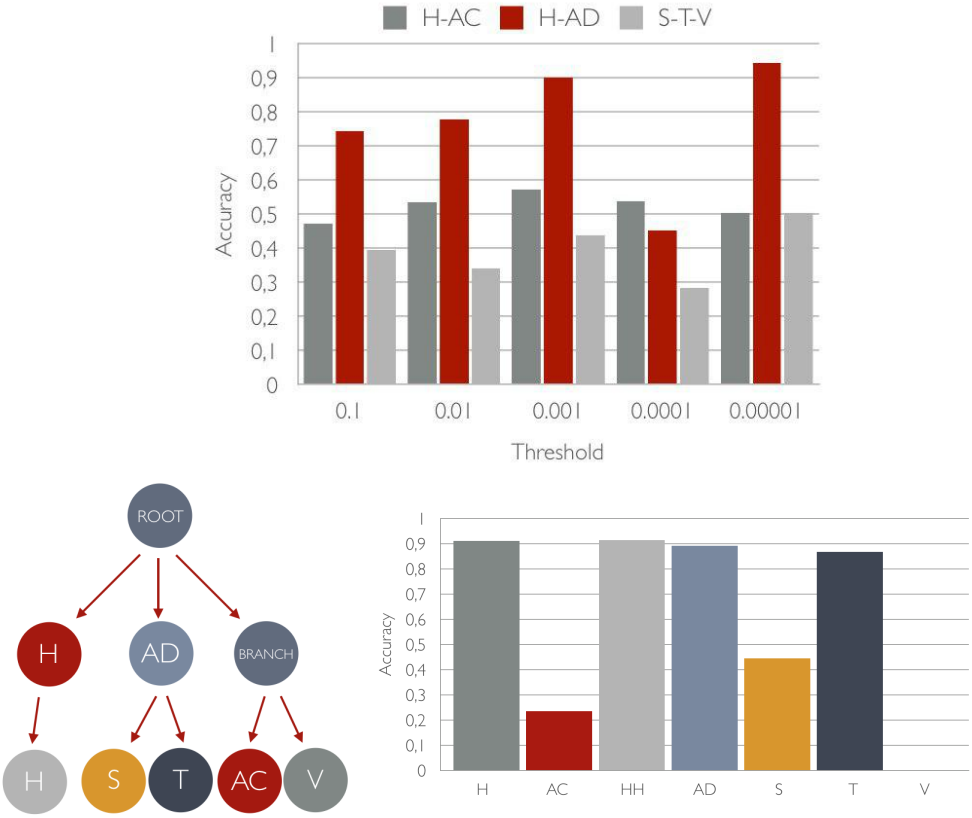

Figure 3: Outcomes using the single class insertion method with the ttest as an alternative likelihood evaluation. The figure on the top shows the classification accuracies for several choices of t_u. The figure on the bottom-left shows the tree structure corresponding to $t_u = 10^{-3}$ (best value on average), while the one on the bottom-right shows the related accuracies.

pended below H (as expected from a biological point of view) and S-T-V are appended below AC. This behaviour may be explained by the strong relationship between V and AC. By inserting S-T-V as batch, class V drags the whole batch below node AC.

In order to overcome this limitation, we exploited single class insertion using both the standard likelihood (Fig. 2) and the t-test (Fig. 3). Although profoundly changed, the tree structures are again coherent. In Fig. 2 the left branch groups H and S class (healthy and nearly healthy samples), showing a worth of interest relationship between H and S, while the right branch groups

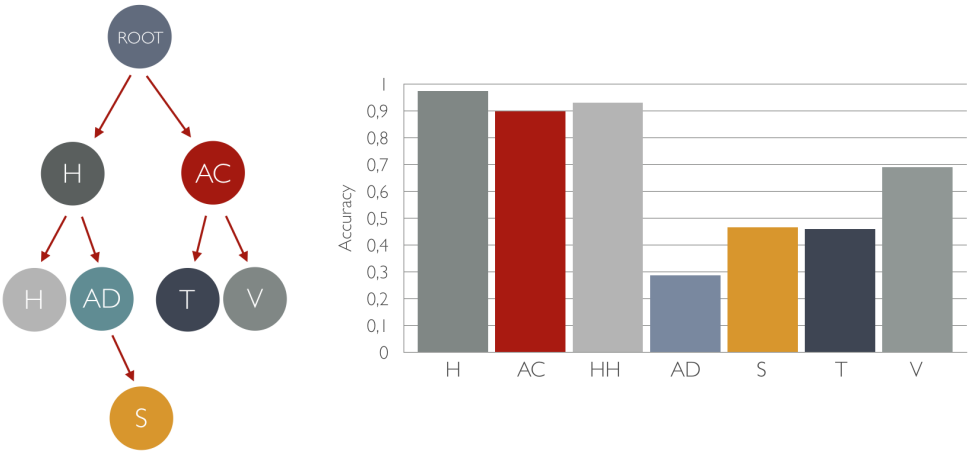

Figure 4: Outcomes using an hybrid insertion (batch insertion for H-AC, H-AD and T-V and single insertion for S). The figure on the left shows the tree structure corresponding to $t_u = 0.6$. The one on the right shows the classification accuracies.

risky or ill situations. We observe an increase in accuracy for the classification of classes related to an Adenoma pathology, although AC accuracy is significantly lower than before. This can be explained by the fact that now class AC is positioned in a lower level and the classification error grows going top-down in the tree.

In Fig. 3, instead, the hierarchy is tripartite: on the left healthy states, in the center nearly healthy and on the right dangerous states. These results are what we expected from the tests done before; as we noticed from the batch insertion tests, there was a strong relationship between AC and V; this relationship can be highlighted here where the algorithm prefers to create a branch node able to distinguish among AC and V instead of flowing V below AD, as expected from a biological point of view and as it does for S and T classes. The lower classification for AC and V can be explained by the fact that the root node's classifier is biased and tend to provide good classification for H and AD while disregarding the data on the branch node. Overall, the best classification accuracies are obtained with single insertion and t-test likelihood. However, trees structures generated with single class insertion seem to vary a lot from run to run. Therefore, we tried an hybrid approach, exploiting human interaction, by inserting H-AC, H-AD and T-V as batches and S as single class. This choice has been taken since by previous experiments we have detected a dissimilarity between S, more near to

the healthy class, and the other two Adenoma classes T and V, more similar instead to the class AC. The algorithm has been performed with different thresholds and the best result is obtained with threshold 0.6 (Fig. 4). As we can see the resulting tree matches our expectations and gives us a good average accuracy of 0.7. However, since batch insertion has been performed without requiring the interaction of the user, we obtain a tree that could classify an image at the same time as AC and T (or V). Even though this is not very accurate from a pure classification point of view, we are able to obtain a good general accuracy for tissues that already present cancer or can present it in the near future. That can alert us of a possible danger and allows to take that sample into consideration fur further analysis.
For what concerns the ANOVA, when we tested it on our data we observed that it tends always to separate the classes and perform a single class insertion. This is due to the fact that our batches of data contains classes enough different from each other that their *between class variance* is not explained by one of the already trained simple models. The ANOVA strategy is probably more useful when the analyzed dataset have a deeper hierarchical structure, such that when trying to classify lower level classes they perform in similar way in the CNNs at the more abstract levels. Finally, we implemented and tested different simple strategies of data augmentation but the improvements in results weren't significantly better than the ones that we already reported.

5. Conclusion and Future Works

In this work we tried to improve the current state-of-the-art of Tree-CNNs. The main novelties are related to:

- the development of flexible ways of growing including batch and single insertions, but also by creating local alternatives to the tree structure;
- the implementation of different techniques for the estimation of the likelihood matrix exploiting both neural and statistical properties;
- the integration of AI and human know-how in the construction of the hierarchical model.

The computation of the likelihood matrix based on a t-test approach allows us to overcome the problems related to a fixed user-defined threshold. In the t-test implementation, in fact, each element of the matrix is computed based

only on a comparison between two samples of scores and does not have any relationship with the other values on the matrix. With the previous computation, instead, the values of each line of L were normalized by the softmax function, so that with an increase in the number of children of a node also the values of L became smaller and the chances of exceeding the threshold decrease significantly, thus causing a breath growing of the tree instead of a depth one. This aspect does not affect substantially the experiments with our data, but can help to generalize the algorithm to bigger datasets.
However, further improvements are needed, especially in the following areas:

- the optimization of base estimator hyperparameters which may change with regard to the analyzed data set;
- the development of more advanced techniques of data augmentation;
- the heuristic optimization of the new designed parameters, such as a threshold that adapts based on the conformation of the tree or a better automatic management of batch and single class insertion.

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, CoRR abs/1409.4842 (2014).

[3] D. Roy, P. Panda, K. Roy, Tree-cnn: A deep convolutional neural network for lifelong learning, CoRR abs/1802.05800 (2018).

[4] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2014).

[6] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.