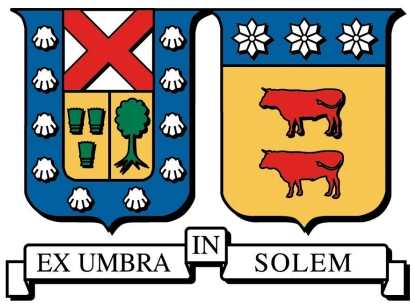
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



Una Arquitectura Basada en Atención para Clasificación Jerárquica con Redes Neuronales Convolucionales

Iván Andrés Pizarro Quezada

Tesis para optar al Grado de
Magíster en Ciencias de la Ingeniería Informática

Profesor Guía: Ricardo Ñanculef A., Ph.D.
Profesor Co-Guía: Carlos Valle V., Ph.D.
Profesor Correferente Interno: Hernán Astudillo R., Ph.D.
Profesor Correferente Externo: Ignacio Araya Z., Ph.D.
Presidente Comisión: Hernán Astudillo R., Ph.D.
22 de julio de 2023

Acknowledgements

I would like to express my most sincere thanks to my professors for their dedication and feedback during this work. To the Chilean Navy for giving me the opportunity to specialize in the area of informatics. And to my family that without their constant support, my studies would not have been possible.

Abstract

This study deals with classification problems in which the class labels form a hierarchy, from broad concepts to more specific categories. Convolutional Neural Nets (CNNs) that use a specialized branch per hierarchy level have become a popular approach for this task in computer vision and other areas. However, inter-level classification consistency is still a problem: the classes predicted at different levels often do not respect the class-subclass constraints encoded by the hierarchy.

Different communication patterns between branches have arisen in the literature to overcome this limitation. This work presents a simpler and more flexible approach: let the neural net decide how branches must be connected. We achieve this by formulating an attention mechanism that dynamically determines how branches influence each other during training and inference.

The hypothesis underlying this research is that adding an attention mechanism to combine intermediate representations extracted from different depths of a convolutional neural network would improve the model's performance in hierarchical classification problems.

Experiments on image classification benchmarks show that the proposed method can outperform state-of-the- art models in terms of hierarchical performance metrics and consistency. Furthermore, although sometimes we found a slightly lower performance at the deeper level of the hierarchy, the model predicts much more accurately the ground-truth path between a concept and its ancestors in the hierarchy. This result suggests that the model does learn not only local class memberships but also hierarchical dependencies between concepts, confirming the associated hypothesis.

Resumen

Este estudio, trata con problemas de clasificación donde las etiquetas forman una jerarquía, de conceptos más generales a categorías más específicas. Para abordar este problema, se ha vuelto popular el uso de Redes Neuronales Convolucionales (CNN) que utilizan ramas especializadas por cada nivel de la jerarquía. Tanto para tareas del área de Visión Computacional como en otras. Sin embargo, la consistencia de la clasificación sigue siendo un problema: las clases predichas en diferentes niveles a menudo no respetan las restricciones de clase-subclase codificadas por la jerarquía. Han surgido en la literatura distintos patrones de conectividad entre ramas para tratar con esta limitación. Nosotros proponemos un enfoque más simple y flexible: dejar que la red neuronal decida como se deben conectar dichas ramas. Lo anterior se logró formulando un mecanismo atencional que determina dinámicamente como las ramas se influencian entre ellas durante el entrenamiento e inferencia.

La hipótesis asociada, es que la introducción de un mecanismo atencional para combinar representaciones intermedias extraídas de distintas profundidades de una red convolucional, permitirá mejorar el desempeño del modelo en problemas de clasificación jerárquica con respecto a modelos presentes en el estado del arte.

Los experimentos realizados con datasets ampliamente utilizados para clasificación de imágenes, demostraron que el modelo propuesto puede superar el estado del arte en términos de métricas de rendimiento jerárquicas y consistencia. Además, pese a que lo anterior genera en algunos casos un rendimiento ligeramente inferior en el nivel más profundo de la jerarquía, el modelo predice con mucha más precisión la relación entre un concepto y sus ancestros. Este resultado sugiere que el modelo no solo aprende las pertenencias a clases locales, sino también las dependencias jerárquicas entre conceptos, lo que confirma la hipótesis estudiada.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many pattern classification applications, class labels can be organized into a hierarchical taxonomy [44, 18]. For example, to help shoppers find products easily, e-commerce platforms such as Amazon maintain a detailed product taxonomy in which categories (e.g., Electronics Bags & Cases) branch into more specific sub-categories (e.g., Laptop Computer Briefcases). Other applications in which hierarchies of this type arise include sentiment analysis [32], web content categorization [12], disease detection [2], and gene function prediction [7, 10]. Hierarchical classification methods are devised to organize data into the hierarchy while respecting and exploiting the class relationships encoded by the taxonomy.

Many methods exist that adapt traditional machine learning algorithms for hierarchical classification. They include top-down approaches, which train different classifiers per node or level of the hierarchy [51], and global approaches, which train a single classifier for the entire hierarchy [21, 22]. Unfortunately, only a few approaches address this task using deep learning methods such as convolutional neural nets (CNNs) [27, 8]. As these models become state-of-the-art in more and more classification problems [37], methods to embed class hierarchies into deep architectures may become increasingly useful and necessary.

The popular approach of augmenting CNNs to include *branches* that support hierarchical classification was first introduced in [57] and then extended in [59]. In this model, referred to as Branch Convolutional Neural Network (B-CNN), a *branch* is a fully-connected subnet that receives a feature representation from a main convolutional block and computes predictions for a specific hierarchy level. The main advantage of this approach is the implementation's simplicity and generality: it can accommodate any standard architecture as the central building block. The main disadvantage is that predictions per level are often inconsistent with the hierarchy: the class predicted for a given level may not be an ancestor of the classes predicted for lower levels. This limitation has motivated many refinements of the original model. For instance, [19] proposed an architecture in which skip connections allow hidden features from a branch to propagate directly to the next branch. This way, the prediction for a given level is explicitly conditioned on the previous one. More

recently, [58] proposed to connect a branch with all the previous branches in the network. Finally, in [23], the authors used a similar connectivity pattern, but instead of propagating hidden representations, they propagate probability estimates from each branch. Although all these methods slightly improve hierarchical classification metrics, we empirically show that their improvements often came from the more specific level of the hierarchy only. Therefore, these models can still produce many predictions inconsistent with the hierarchy at inference time. This result shows that current CNNs do not accurately learn hierarchical class relationships.

This work highlights two limitations of current CNNs for hierarchical classification. First, existing approaches are limited to top-down connectivity patterns: features propagate unidirectionally from the top to deeper branches. This way, predictions at detailed levels of the class hierarchy do not explicitly give feedback to the upper levels. In addition, the connectivity pattern is static; it cannot change during training or inference. We hypothesize that a more flexible communication pattern can improve the ability of CNNs to learn hierarchical relationships and, thus, hierarchical consistency.

Inspired by recent advances in deep learning, this work proposes BA-CNN, a method that extends the original B-CNN model by connecting its branches through an attention mechanism. The main motivation for our method is that Attention [48] currently allows models to selectively and dynamically aggregate information from different parts of the computational graph with a minimal computational burden. Thus, by connecting B-CNN's branches through an attention mechanism, we allow a branch to condition its predictions on context vectors of the whole class hierarchy without cumbersome hand-crafted connections. This mechanism breaks the static top-down architecture of current models. In particular, fused vectors can include features from lower and deeper branches and, therefore, from coarser and deeper levels of the class hierarchy. Moreover, by adjusting the attention weights, attention allows the model to change how branches influence each other during learning and prediction.

Experiments on image classification benchmarks show that the proposed method can outperform state-of-the-art models in terms of hierarchical performance metrics, sometimes at the cost of slightly lower performance on the most specific level. Furthermore, the model's predictions reconstruct more accurately the ground-truth path between a concept and its ancestors in the hierarchy, which suggests the model does learn not only class memberships but also their dependencies. In addition, we show that BA-CNN has a memory footprint and training time close to that of a flat CNN and lower than that of more explicit and dense connectivity patterns for hierarchical classification.

The contribution of this thesis is threefold:

- First, proposing a novel extension of the B-CNN model [59] for deep hierarchical classification. The model uses an attention module that allows feature maps to flow in different directions of the hierarchy, top-down *and* bottom-up.

Moreover, the attention mechanism determines how branches influence each other in a dynamic and data-driven way.

- Second, extend the actual comparison in the literature by including four current branched CNN architectures for hierarchical classification, three benchmark datasets, and four different metrics.

- Third, proposing two custom metrics named hierarchical accuracy and hierarchical consistency. Those metrics complement the chosen state-of-the-art metrics and help to measure the ability of the model to learn hierarchical dependencies between concepts more directly.

1.1 Hypothesis

The hypothesis underlying this study is that adding an attentional mechanism to combine intermediate representations extracted from different depths of a convolutional neural network will improve the model's performance in hierarchical classification problems with respect to B-CNN, H-CNN, Add-net, and Concat-net.

1.2 Objectives

The general objective of this work is to create an attentional mechanism that allows a CNN to solve hierarchical image classification problems competitively compared to the current state of the art.

The specific objectives associated with this work are the following:

1. Design, implement and evaluate at least one attentional mechanism that allows combining intermediate representations extracted from different depths of a convolutional network to solve hierarchical classification problems. This is addressed in Chapters 4 and 5.

2. Generate a free code *framework* that allows reproducing state-of-the-art methods and comparing their results with the proposed methods. This is publicly available at `https://github.com/IvanPizarroQ/BA_CNN`.

3. Select for experimentation at least three datasets previously used to validate hierarchical classification models in images. This is addressed in Chapter 5.

4. Compare the results of the proposed models with other state-of-the-art methods considering the accuracy obtained at the different levels of the class hierarchy. This is addressed in Chapter 5.

5. Evaluate the consistency of the model's predictions with respect to the class hierarchy used. This is addressed in Chapter 5.

1.3 Structure

The present work is structured as follows. Chapter 2 provides a brief background referring mainly to the concept of hierarchical classification, neural networks, and the technical background of attention mechanisms relevant to our formulation. Chapter 3 reviews related work on methods to train deep CNNs on class hierarchies. Our architecture is motivated and described in Chapter 4. Chapter 5 explains the experimental setting and results, considering hyperparameter tuning, comparisons between the proposed method with four other methods, and analysis of attention weights and time-space complexity. Finally, we summarize our conclusions and discuss future research directions in Chapter 6.

Chapter 2

Background

2.1 Hierarchical Classification

In pattern recognition, a classifier is a mapping $f : X \to Y$ between data domain X and a finite set of class labels C, representing different concepts in an application domain. Classification algorithms often assume classes are unrelated. Hierarchical classification instead copes with problems in which the classes Y organize into a *taxonomy* [44] that agglomerates classes to create more abstract concepts.

Wu et al. [54] defined such a taxonomy as a tree-structured traditional concept hierarchy defined over a partial order set $(C;\prec)$, where $\prec$ represents the "is-a" relationship. Silla and Freitas [44] defined the "is-a" relationship as asymmetric, antireflexive, and transitive:

1. The only one greatest element is the root of the tree R, i.e., $\exists! R \in C : \forall c \in C, c \neq R \Rightarrow c \prec R$.

2. $\forall c_i, c_j \in C$ if $c_i \prec c_j$ then $c_j \not\prec c_i$.

3. $\forall c_i \in C$, $c_i \not\prec c_i$.

4. $\forall c_i, c_j, c_k \in C, c_i \prec c_j$ and $c_j \prec c_k$ imply $c_i \prec c_k$.

On the one hand, the existence of a taxonomy for the classes of interest represents an opportunity to improve classification performance. As concepts at the lower levels of the hierarchy are a specialization of concepts at the upper levels, it should be easier to classify an instance by following the taxonomy. The classifier can first discriminate between simpler/coarser concepts, discarding a large set of possible outcomes, and then focus on the specific features that make the instance part of a sub-category. For example, knowing that *birds* cannot be *felines* can help the classifier discard feline-specific features when the *bird* category is much more likely than the *mammal* category. On the other hand, the existence of a taxonomy presents a learning challenge. If an instance belongs to a given class, it automatically belongs

to all its superclasses, and the classifier should respect this *hierarchy constraint*. For example, if the model classifies an instance as a *feline*, it should classify the instance as a *mammal* and not as a *bird*.

Methods to leverage and respect concept hierarchy differ from three main criteria [46], [13]. The first is the hierarchical structure used, either a tree or DAG, as depicted in Fig. 2.1. The main difference between using a DAG instead of a tree is that a node can have more than one parent node in the DAG (a category may belong to multiple superclasses). The second criterion is the completeness of the decisions made by the classifier, i.e., whether the method can stop the classification at any node of the class hierarchy (non-mandatory leaf-node prediction) or is constrained to stop at a leaf (mandatory leaf-node prediction). The third criterion is related to how the classifier explores the hierarchical structure. The most straightforward approach ignores the class hierarchy, typically predicting only classes at leaf nodes. Methods in this category are called *flat classifiers*. Another way to achieve this is to learn the whole class hierarchy using a single classifier. Methods in this category are called *global* (or big-bang) classifiers. A third possible scenario involves learning the taxonomy class using a set of local classifiers. These methods are also known as top-down classifiers [44].

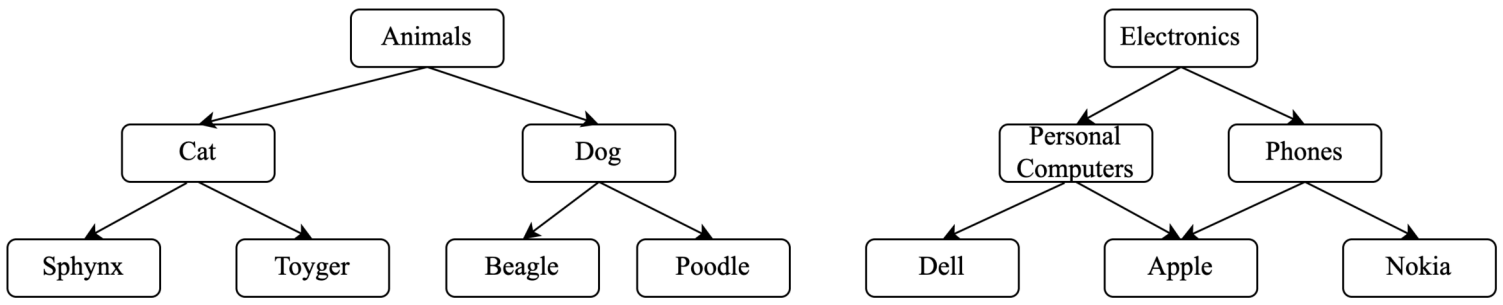


Figure 2.1: Hierchical classification structures: Tree (left) and DAG (right).

2.1.1 Flat Classifiers

This method is also known as the direct approach [9], or bottom-up approach [6]. It completely ignores the class hierarchy, typically predicting only the classes at leaf nodes. Therefore, this approach acts as a traditional classification algorithm during training and inference (testing). Despite neglecting the hierarchy, this approach can indirectly lead to hierarchical classifications in some cases. Indeed, assuming the "is-a" relationship, we can assign to an instance all the ancestor classes corresponding to the leaf class predicted by the model. It is worth noting that the latter applies only to a tree hierarchy and if the taxonomy is perfectly known. To illustrate this approach, Fig. 2.2 shows the use of a flat multi-class classification algorithm.

The main disadvantage of flat classifiers is that they do not exploit possible correlations between the target concepts, which help the learner to improve generalization such as multi-task learning and transfer learning benefit from learning inter-related

tasks jointly [28]. Furthermore, theoretical studies have revealed that neglecting the hierarchical structure is especially unfavorable in large-scale multiclass scenarios, where a flat classifier must simultaneously discriminate between many unbalanced categories [4].

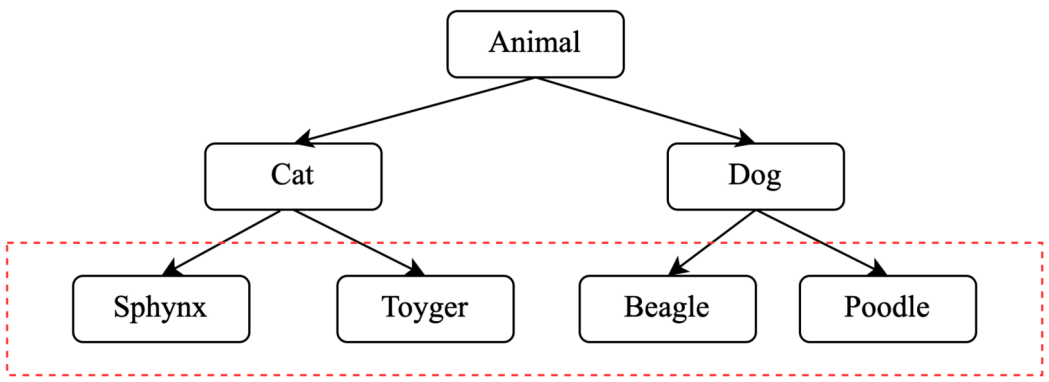


Figure 2.2: Local classifier approach using a multi-class classification algorithm.

2.1.2 Local classifier

These approaches handle the hierarchy using a local top-down approach. For a new example in the test set, the algorithm first predicts the coarsest level and then uses this prediction to narrow the choices at the following (finer) level. This procedure is applied recursively until the classifier reaches a leaf node. The major drawback of this approach is that misclassification at a given class level propagates to the lower levels of the class hierarchy. According to Silla and Freitas [44], there are three main approaches to implementing a classifier of this type:

1. Local classifier per node.
2. Local classifier per parent node.
3. Local classifier per level.

The local classifier per node approach is the most used in the literature, and it consists of training a binary classifier for each node of the class hierarchy except the root. Figure 2.3 illustrates this approach. For training, there are several ways to define each binary classifier's positive and negative examples. Then, for inference, each classifier predicts whether or not an example belongs to the class corresponding to the actual node. An advantage of this approach is that it allows assigning more than one label per level, as is required for multi-label hierarchical classification problems. However, a significant disadvantage of this method is that prediction errors in the nodes can generate inconsistent predictions, in other words, predictions that do not respect the class taxonomy. Therefore, due to the above, methods with this approach must have some way of correcting inconsistencies.

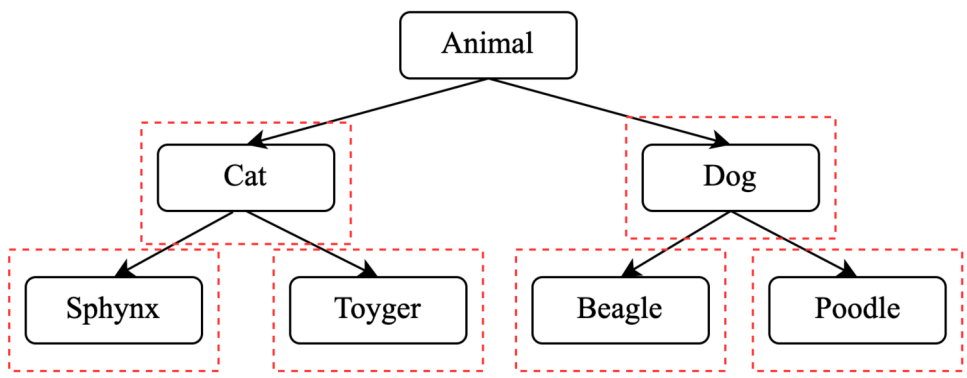


Figure 2.3: Example of the local classifier per node approach.

The local classifier per parent node approach is illustrated in Figure 2.4. This approach implements a multi-class classifier per parent node, trained to distinguish between its child nodes. It is worth mentioning that this method is also prone to inconsistencies if a post-processing method to correct the predictions is not applied.

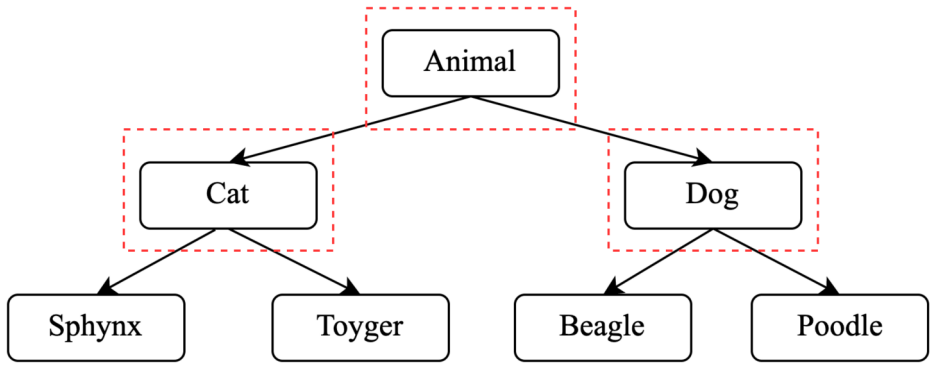


Figure 2.4: Example of the local classifier per parent node approach.

Finally, the local classifier per level approach consists of implementing a multi-class classifier per level of the class taxonomy. Fig. 2.5 illustrates this idea.

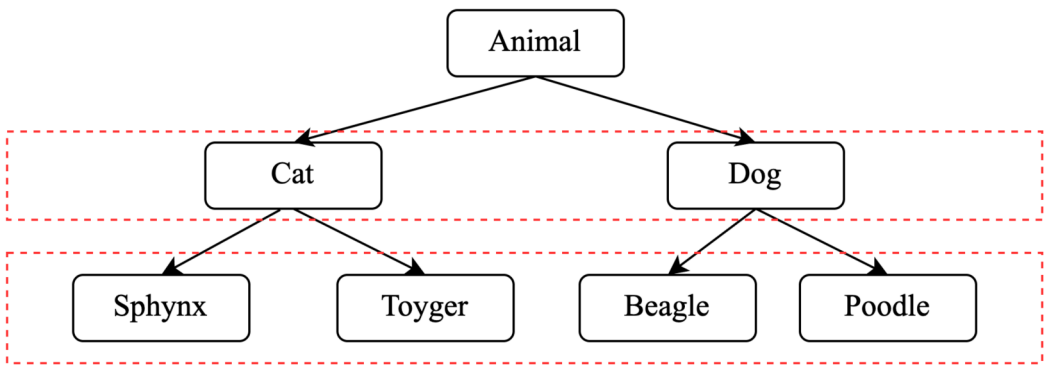


Figure 2.5: Example of the local classifier per level approach.

2.1.3 Global classifier

As Fig. 2.6 shows, algorithms in this category train a single (usually complex) classification model to learn the class hierarchy. For example, Labrou and Finin [30] pro-

posed a text-mining classifier to address *Yahoo!*'s hierarchical categories. It learns the hierarchy using a set of topic prototypes and a classification method resembling Rocchio's document categorization approach [38]. For inference, this method first computes the similarity of a test document with each topic. The method then classifies the document in the corresponding topic if the similarity is beyond a certain threshold. Kiritchenko et al. [21] [22] considered the hierarchical class problem a multi-label classification problem. During the training process, the method expands the label set of all the training examples with their corresponding ancestor labels and treats them as different possible outcomes. This approach is naturally prone to hierarchical inconsistencies, so the authors considered a post-processing stage that considers all outputs to ensure that the hierarchical constraints are respected.

It is worth noting that the output of a global classifier might be easier to interpret than that of a local classifier because the complexity of the decision procedure implemented by the former is often lower. For example, the experiments reported in [49] showed that the number of rules generated by the global approach was much smaller than the number of rules generated by the local approach. In addition, the global classifier approach does not suffer from the significant drawback of the local classifier approach, namely, the fact that a misclassification at a given class level is propagated to the lower levels of the class hierarchy.

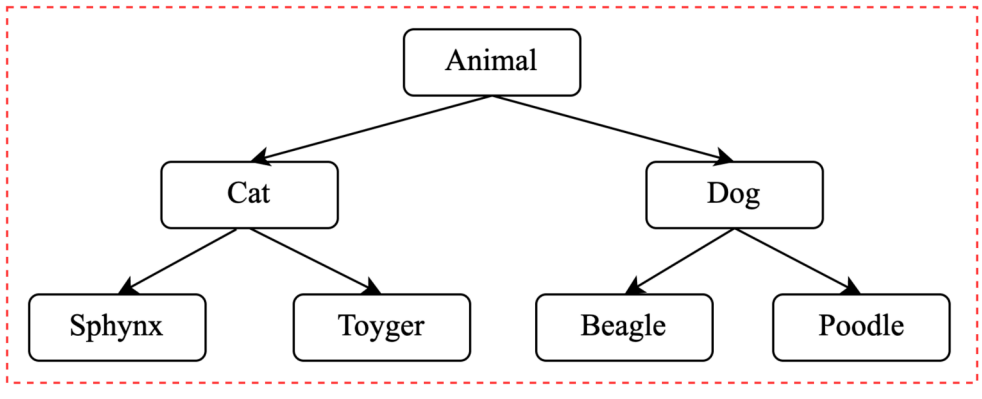


Figure 2.6: Example of the global classifier approach.

2.2 Hierarchical consistency

We expect that a hierarchical classification algorithm produces predictions that respect class hierarchy. This property is referred to as *hierarchical consistency*. Wu et al. [54] defines it as: "A label set C_i assigned to an instance d_i is called consistent with a given hierarchy if C_i forms a connected proper subgraph of the hierarchy graph rooted in the top node." For example, in the hierarchy in Fig. 2.7, the correct label set for the instance $d_1 = Nokia$ is {"Electronics", "Phones", "Nokia"}. The label set {"Electronics", "Phones", "Dell"} is called a hierarchical inconsistency or a class-membership inconsistency [44].

As the hierarchical structures are usually Tree or DAG, we can exclude the root node from any ancestor set since it does not provide additional information.

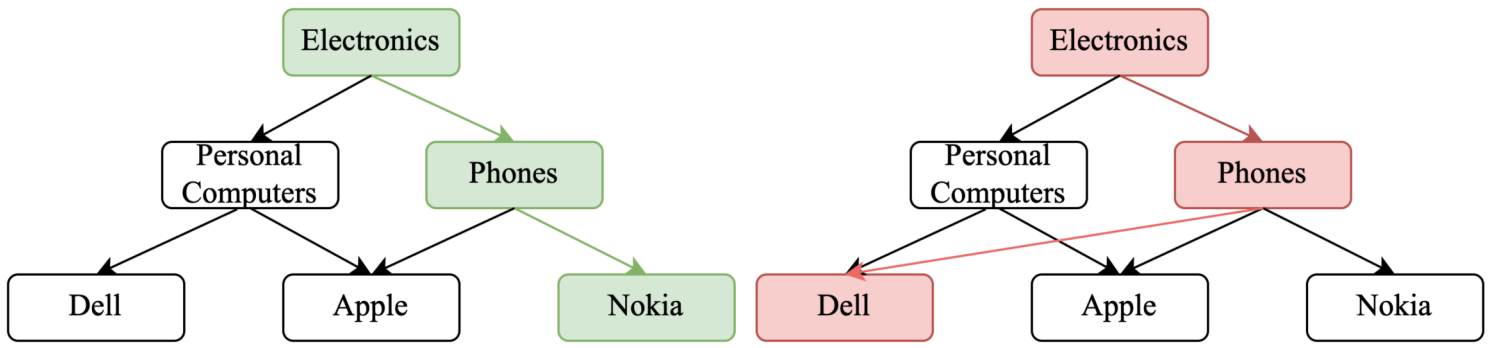


Figure 2.7: Example of a hierarchical consistent label path (left) and a hierarchical inconsistency (right).

2.3 Neural Networks

Machine Learning (ML) is the study of algorithms that can learn from experience with respect to some task. This task is formally a function f^* that maps input data x to output data y. For example, for a classification task, $y = f^*(x)$ maps an input x to a category y. As f^* is unknown, the ML model defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that results in the best function approximation. In the learning process, these parameters are adjusted to optimize a *objective function* that measures the model's performance. It said that the machine is learning if it can improve his performance using the experience or data.

Neural Networks, also called Artificial Neural Networks (ANNs), are a family of ML algorithms with a structure inspired by the human brain. As depicted in Figure. 2.8, they are composed of node layers, tipically: an input layer, one or more hidden layers, and an output layer. Feedforward neural networks [16], also called Deep feedforward networks or multilayer perceptrons (MLP), are ANNs that approximate f^* using a computational graph in which the information flows in only one direction without loops.

Like other ML methods, neural networks can be trained to learn a task by optimizing their layer's parameters according to an objective function. *Gradient descent* (GD) is a commonly-used algorithm for optimization. This algorithm uses the gradients of the objective function with respect to the parameters to adjust them until the objective function converges to a minimum. As the computation of partial derivatives may become impractical for complex networks, efficient implementations of GD are required. Rumelhart et al. [39] proposed an algorithm called *Backpropagation* that computes the gradient via the chain rule. This method allows computing the gradient layer by layer to avoid redundant computation.

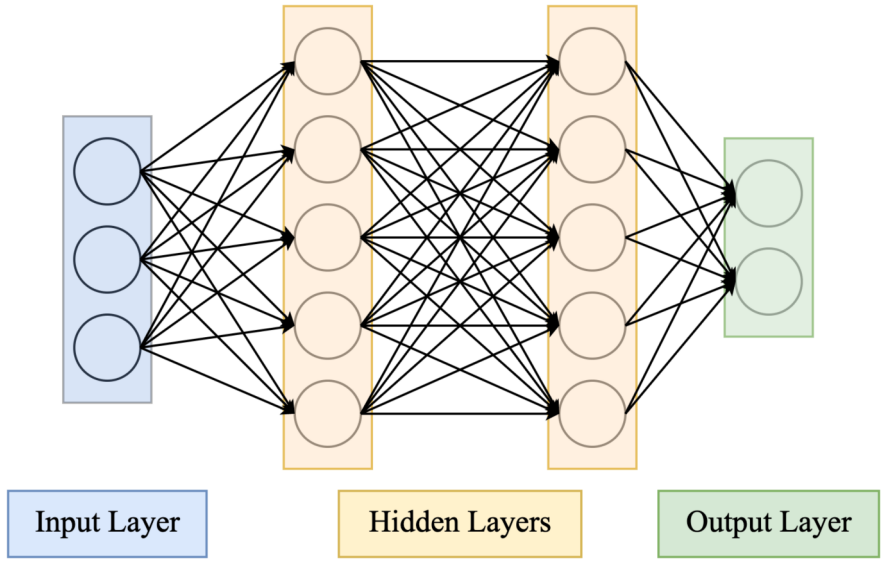


Figure 2.8: General structure of an ANN.

2.3.1 Convolutional Neural Networks

Images are commonly represented as a two-dimensional grid of pixels. Each pixel corresponds to one or multiple numerical values. As fully connected MLP required to be fed with a one-dimensional vector, the standard approach to deal with images was flattening the images, irrespective of the spatial relation between pixels. To deal with this limitation, LeCun et al. [31] presented the convolutional neural networks, a family of neural networks designed for working with images.

CNN layers perform convolution operations on an image (or a feature map) and a filter matrix (also called a filter or a convolution kernel). Fig. 2.9 shows the typical architecture of a CNN, where the input image passes through convolutional layers to obtain feature maps in a stage usually called feature extraction. In this stage, the first convolutional layers may extract some low-level features, and a multi-layer network can extract more complex features based on the low-level ones. After the feature extraction, as shown Fig. 2.9, a fully connected network is generally used as an output of the CNN. Fig. 2.9 illustrates a CNN for classification, a common task in the pattern recognition field.

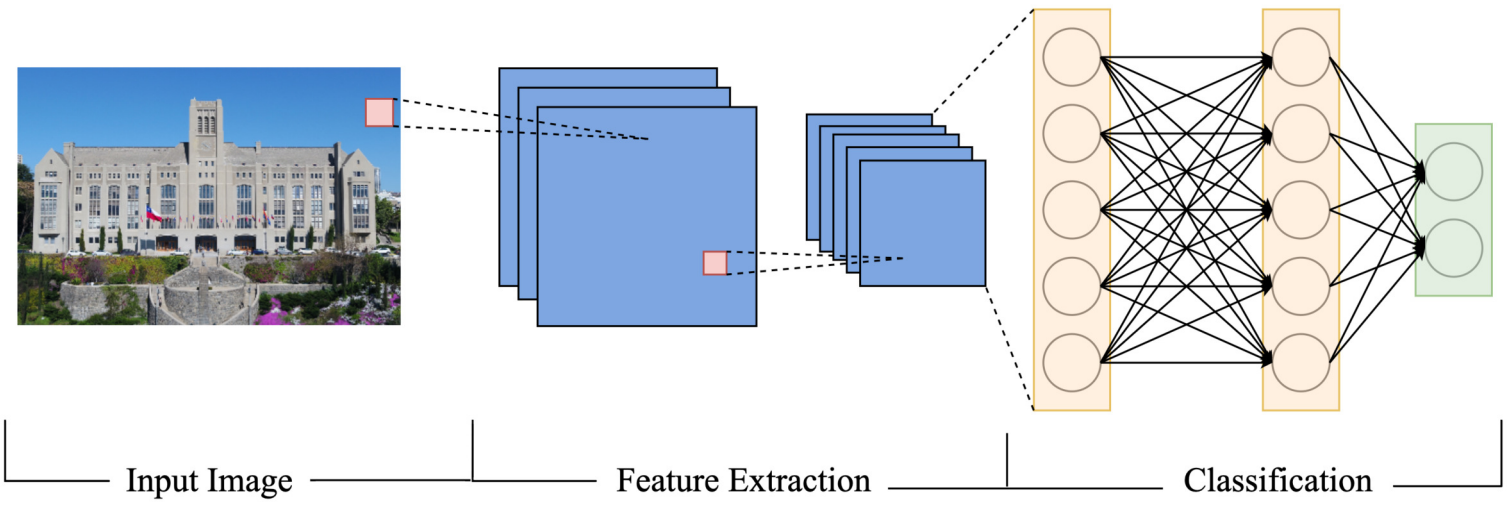


Figure 2.9: Example of a CNN for image classification.

2.3.2 Attention Mechanisms

Attention is a recent technique for improving deep learning models that has shown promising results in Natural Language Processing [48], Speech Recognition [11], and Computer Vision [17]. Inspired by the ability of humans to selectively concentrate on parts of the information when processing large amounts of data, attention mechanisms allow a model to focus on salient features of the input data or its internal representations to solve a task.

Formally, given a sequence of vectors (or more generally tensors) $V = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_M\}$ produced at a given level of the neural net, an attention mechanism computes a new set of feature representations $A = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_N\}$ by recombining the elements of V as follows:

$$\boldsymbol{a}_n = \sum_{m=1}^{M} \alpha(\boldsymbol{q}_n, \boldsymbol{k}_m)\boldsymbol{v}_m \,, \tag{2.1}$$

where $\boldsymbol{k}_m$ is a vector referred to as *a key*, $\boldsymbol{q}_n$ is a vector referred to as *a query*, and $\alpha(\boldsymbol{q}_n, \boldsymbol{k}_m) \in [0, 1]$ is the *attention weight* assigned by the attention mechanism to *the value* $\boldsymbol{v}_m$. In this abstraction, the query encodes an information need or task, while the key $\boldsymbol{k}_m$ is a descriptor of the information contained in $\boldsymbol{v}_m$. The attention weight $\alpha(\boldsymbol{q}_n, \boldsymbol{k}_m)$ determines the relevance of the information source $\boldsymbol{v}_m$ for the task $\boldsymbol{q}_n$ and is computed by applying the Softmax function to the scores vector ξ_{nm}. For instance, the values $\boldsymbol{v}_m$ may correspond to the latent vectors computed by an encoder recurrent neural network (RNN) fed with input text, and the queries $\boldsymbol{q}_n$ may correspond to the latent vectors computed by a decoder RNN attempting to translate the text into another language. The keys may correspond to the values themselves or their projection into a space in which the languages can be more easily aligned. In this example, α_{nm} quantifies the attention that the model needs to give the m-th input word to predict the n-th output word.

Different attention models differ in the implementation of the scoring function

that computes ξ_{nm}. Bahdanau et al. [5] proposed an additive attention mechanism in which the scores vector were first computed as

$$\xi_{nm} = \tanh\left(W^{(1)}\boldsymbol{q}_n + W^{(2)}\boldsymbol{k}_m\right) , \tag{2.2}$$

were $W^{(1)}$ and $W^{(2)}$ are learnable matrices. Then, a Softmax function is applied to obtain positive attention weights that sum to 1. Later, Luong et al. [35] proposed a multiplicative attention mechanism in which the scores vector are computed as

$$\xi_{nm} = \boldsymbol{q}_n W \boldsymbol{k}_m , \tag{2.3}$$

where W is a learnable matrix. It is noteworthy that, in these attention models, the keys coincide with the values. Key/value/query abstraction became popular after the introduction of Transformers [48], which came with the concept of multihead dot-product attention. This mechanism extends the multiplicative model of (2.3), with two significant improvements. First, the keys, values, and queries are projected onto a subspace using the learnable matrices $W^{(k)}$, $W^{(v)}$, and $W^{(q)}$. Second, multiple attention mechanisms or *heads* can operate in parallel. Finally, the results of these K *heads* are concatenated and projected onto the desired dimensionality using a learnable matrix $W^{(o)}$.

Applying the attention mechanism to internal network representations has led to the concept of *self-attention*, a method that allows some models to learn from sequences without recurrent connections [48]. Recent research has also shown that attention is a powerful method for improving the interpretability of deep-learning models [3]. Indeed, if the mechanism is strategically placed to attend to the input or internal features with clear semantics, visualizing the attention weights can help humans to interpret the model's predictions [33].

Chapter 3

Literature Review

Yan et al. [57] were the first to introduce the idea of extracting information from the hierarchical structure of a Convolutional Neural Network to enrich the prediction quality. The authors proposed the Hierarchical Deep Convolutional Neural Network (HD-CNN) for image recognition. This approach uses a specialized CNN for image recognition as the base (building block). Next, it uses a classifier for the coarse level that contributes to the finest level. One disadvantage of this method is that its architecture only accepts a hierarchy of two levels. Subsequently, Zhu and Bain [59] presented the Branch Convolutional Neural Network (B-CNN). They were inspired by the idea that the first layers of the CNN obtain information from higher-level levels of the hierarchy. As shown in Fig. 3.1, this method uses an existent convolutional net (e.g., VGG16) as a central feature extractor (common backbone) and generates many branch neural networks as levels have the class hierarchy. Each branch neural network is a feedforward neural network that predicts a given level of the hierarchy. Next, a final loss function computes the weighted sum of the loss functions of each branch. They also presented a training method for this type of structure called *Branch Training* Strategy (BT-Strategy). It consists of modifying the weights of the branch losses while training the network. Thus, to improve performance, a weight update process is performed from the lower-level parameters to the higher-level parameters.

Various authors have implemented the B-CNN to solve different problems. For example, Seo and Shin in [42] applied the B-CNN model in the context of retail with the Fashion MNIST dataset, grouping the ten classes of the dataset into six superclasses and then, in turn, grouping these superclasses into two classes. To implement this, the authors used VGG16 and VGG19 as the central feature extractor. In addition, Sali et al. [40] employed the B-CNN model to classify gastrointestinal disorders on histopathological images using a two-level hierarchy.

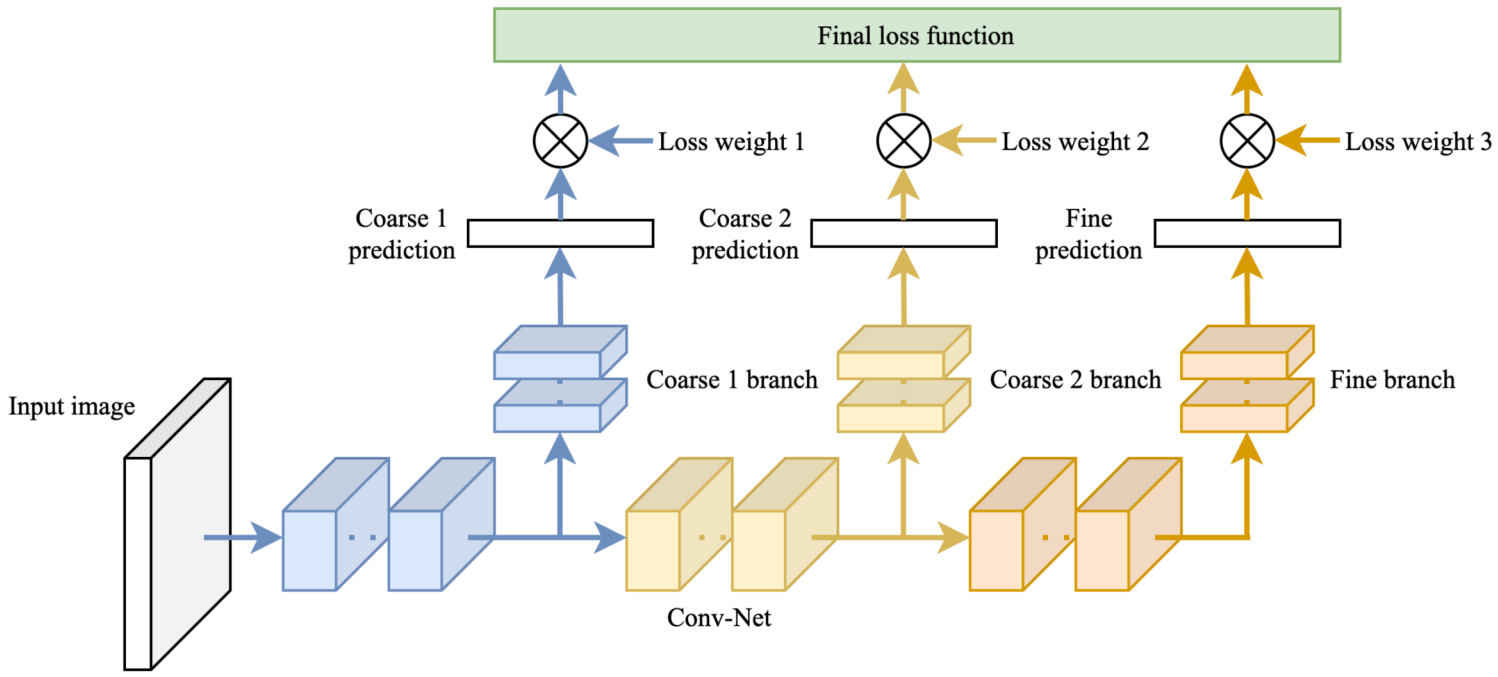


Figure 3.1: B-CNN architecture.

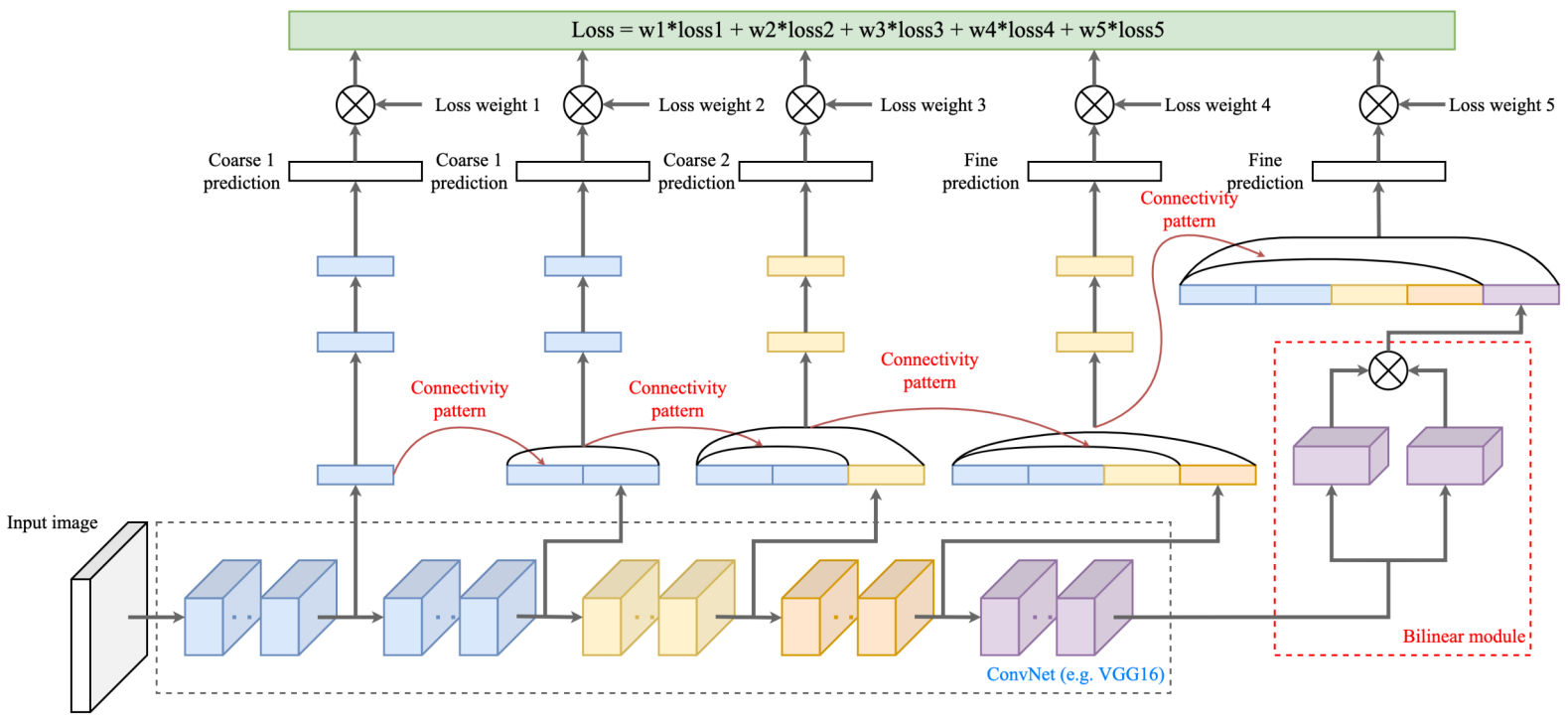


Figure 3.2: HB-CNN architecture.

To that moment, branches were treated independently, but authors such as Inoue et al. [19] and Zhang et al. [58] demonstrated the benefit of interconnecting branches. Indeed, they showed that branches can complement information from different hierarchy levels during training. Inoue et al. proposed the so-called Concat-net and Add-net, which consists of connecting the last dense layer of the branches in a top-down manner. For example, the second branch would receive the last activation tensor from branch 1, and branch three would receive that from branches 1 and 2. This interconnection concatenates these layers for the Concat-net model and adds them in the Add-net model, demonstrating advantages over the approach of using the branches independently (B-CNN). Zhang et al., similar to Inoue et al., also pro-

pose to connect the branches in a model called Hierarchical Bilinear Convolutional Neural Network (HB-CNN). As shown in Fig. 3.2, this was performed from the first dense layer of the branch. This connection is called Connectivity Pattern (CP), and the authors showed that it is beneficial not only to extract one branch per level but it can be more than one; in their experiments, they extracted three and five branches. Furthermore, it uses the Bilinear CNN (B-CNN) proposed by Lin in [34] to provide further enhancement to the fine level. The authors presented HB-CNN without the Bilinear CNN as Hierarchical Convolutional Neural Network (H-CNN). All the aforementioned authors applied the BT-Strategy for training, which entails an exhaustive process of adjusting the weights of the different loss functions and estimating the number of times to apply this change.

All works mentioned above present two main limitations. First, features between branches propagate unidirectionally from coarse to fine levels, causing the fine level to benefit most with this connection and not consider bottom-up feedback. The second limitation is that the connectivity patterns between branches can not change during training or inference, so it has to be carefully tuned and assuming that said interconnection is beneficial for all classes, denying the possibility that for some, the most beneficial connection is different.

Other Deep Learning approaches presented in the literature do not exploit the hierarchical structure of the features extracted by the CNN, i.e., they do not make correspondences between levels of the CNN and levels of the hierarchy to enrich the quality of predictions. For instance, Kolisnik et al. [23] trained a modified VGG16 architecture with teacher forcing, using the true labels of a higher level to train lower levels. They validated their results on the Kaggle Fashion Product Images data set [1]. La Grassa et al. [29] proposed an architectural extension that could be adapted to generic neural networks. They chose a base model and added a set of neural layers equal to the number of levels in the hierarchy. From the base model, they compute a Center Loss function [52] and a cross-entropy loss function for each added layer. Finally, they compute a total loss function by adding all losses. In the medical field, Kowsari et al. [25] proposed the Hierarchical Medical Image classification (HMIC) model that uses one CNN architecture for each parent node in the hierarchy. They validated this idea using a medical dataset in which the first level has three classes (Normal, Environmental Enteropathy, and Celiac Disease), and the child level of Celiac Disease is based on severity (I, IIIa, IIIb, and IIIc). Gao et al. [15] proposed a deep hierarchical classification framework, tested on text and images, composed of three parts: a Feedforward Neural Network, a Hierarchical Embedding Network (HEN), and a Hierarchical Loss Network (HLN). The first is to obtain a root representation used in the HEN to compute an independent representation per level; the second is composed of a representation per level, where the representation of a level is computed by concatenating the representations of all previous levels and the corresponding independent representation. The third consists of a dependence loss to punish if the model does not predict the classes according to the hierarchy

and computes a total loss by adding the losses per level.

Chapter 4

Proposed Model

This chapter presents our method to train a CNN for hierarchical classification. According to Silla's categorization framework [44], the proposed model corresponds to:

- SPP (Single Path Prediction): the model can assign at most one path of the predicted labels to each data instance.
- MLNP (Mandatory Leaf Node Prediction): the model always assigns leaf classes.
- GC (Global Classifier): one single model assigns labels for all levels.

In addition, the model does not assume that the hierarchy is a tree or a DAG and works in both cases. Indeed, the method only assumes that the classes form a top-down taxonomy composed of B different levels and does not require knowledge of the precise relationships between concepts of different levels in advance: the method learns these relationships from data.

Building on B-CNN, our model uses a multi-branch architecture that includes a central feature extractor and a series of branches or sub-models devoted to classifying the input data at each level of the hierarchy. As depicted in Fig. 4.1, our main contribution is an extension of B-CNN based on an attention mechanism that allows feature maps to flow in different directions of the hierarchy. In addition, the attention mechanism determines how the branches influence each other in a dynamic and data-driven way.

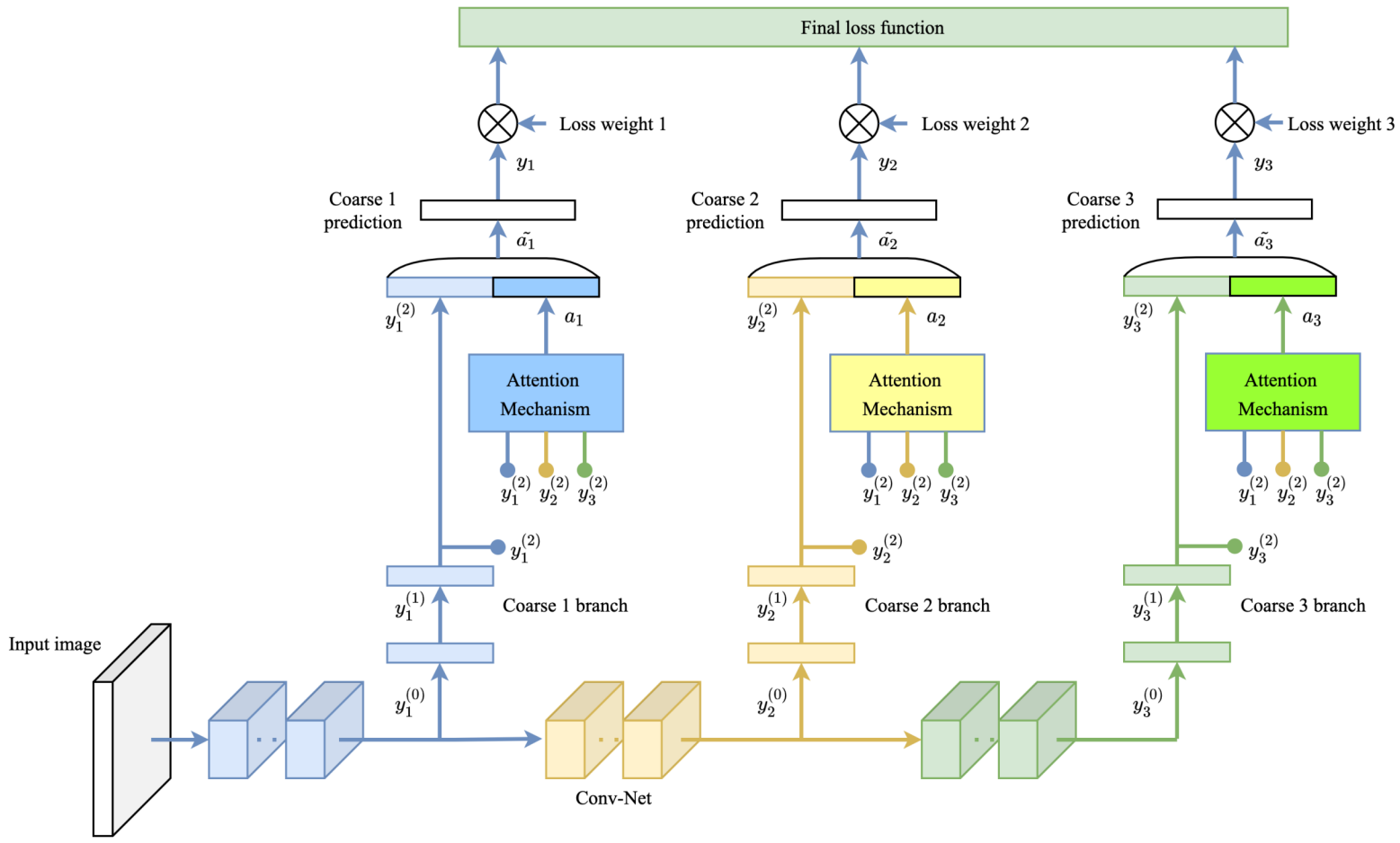


Figure 4.1: Proposed architecture BA-CNN for a 3-level hierarchy.

4.1 Branched Architecture

Formalizing recent approximations, if $\boldsymbol{x}$ denotes a possible input to the system, the central block implements a transformation of the form $\boldsymbol{z}^{(L)} = F(\boldsymbol{x})$ constructed as a composition of L simpler transformations or "layers"

$$\begin{aligned} \boldsymbol{z}^{(\ell)} &= f_\ell(\boldsymbol{z}^{(\ell-1)}) \quad \forall \ell \in [L]\,, \\ \boldsymbol{z}^{(0)} &= \boldsymbol{x}\,. \end{aligned} \tag{4.1}$$

Similarly, each branch implements a layered transformation $\boldsymbol{y}_b^{(K_b)} = G_b(\boldsymbol{x}_b)$ of K_b layers defined recursively as

$$\begin{aligned} \boldsymbol{y}_b^{(\ell)} &= g_\ell(\boldsymbol{y}_b^{(\ell-1)}) \quad \forall \ell \in [K_b]\,, \\ \boldsymbol{y}_b^{(0)} &= \boldsymbol{x}_b \quad \forall b \in [B]\,, \end{aligned} \tag{4.2}$$

where $\boldsymbol{x}_b$ is a tensor specifically prepared for level b. Predictions for each level of the hierarchy can be obtained from a simple fully connected layer fed with $\boldsymbol{y}_b^{(K_b)}$,

$$\boldsymbol{y}_b = \sigma(W_b \boldsymbol{y}_b^{(K_b)} + \theta_b)\,, \tag{4.3}$$

where σ denotes a Softmax activation function. W_b and θ_b are the parameters of the fully connected layer for level b.

A key difference between existing models is how a branch is allowed to condition the prediction of another branch. For example, in [59], the branches were not explicitly connected and depended on each other only through the shared feature extractor. The input of branch b is a feature map extracted from the central block, that is $\boldsymbol{x}_b = \boldsymbol{z}^{(\ell_b)}$ for some $\ell_b \in [L]$ such that $\ell_b > \ell_{b'}, \forall b > b'$. The output of branch b is not independent of branch $b' < b$ because x_b depends on $x_{b'}$, which is a more primitive representation of the input data.

In [23], two contiguous branches get explicitly connected by defining $\boldsymbol{x}_b$ as the concatenation of the previous branch's output with the feature representation drawn from the central block, i.e. $\boldsymbol{x}_b = \boldsymbol{z}^{(\ell_b)} \oplus \boldsymbol{y}_{b-1}^{(K_{b-1})}$. This architecture allows branch b to explicitly condition the prediction of branch $b+1$ and resembles the Jordan recurrent connections used in sequence modelling. In [42], the previous branch's output is substituted by pre-output activations, i.e. $\boldsymbol{x}_b = \boldsymbol{z}^{(\ell_b)} \oplus \boldsymbol{y}_{b-1}^{(k_{b-1})}$ for some $k_b \in [K_b]$. In [58], each branch get explicitly connected to all previous branches by defining $\boldsymbol{x}_b$ as

$$\boldsymbol{x}_b = \boldsymbol{z}^{(\ell_b)} \oplus \left(\bigoplus_{\tilde{b}=1}^{b-1} \boldsymbol{z}^{(\ell_{\tilde{b}})} \right) . \tag{4.4}$$

In [19], branches are connected similary to [58], but instead of connecting the representation extracted from the central block, they connect the pre-softmax layer $\boldsymbol{y}_b^{(K_b)}$. This connection generates a new pre-softmax layer $\tilde{\boldsymbol{y}}_b^{(K_b)}$, defined for Concat-net as

$$\tilde{\boldsymbol{y}}_b^{(K_b)} = \boldsymbol{y}_b^{(K_b)} \oplus \left(\bigoplus_{\tilde{b}=1}^{b-1} \boldsymbol{y}_{\tilde{b}}^{K_{(\tilde{b})}} \right) . \tag{4.5}$$

Add-net uses the same definition, but instead of performing the concatenation, they use the summation.

4.2 Attention Mechanism

Generalizing the above ideas and inspired by recent advances in sequence modelling, we propose replacing these (increasingly complex) connectivity patterns with an attention module. Our aim is to feed branch b using a context vector $\boldsymbol{a}_b$ constructed from representations $\mathcal{B} = \{\boldsymbol{y}_1^{(k)}, \boldsymbol{y}_2^{(k)}, \ldots, \boldsymbol{y}_B^{(k)}\}$ drawn from all the other branches in the network, coarser and finer ones. To this end, we propose an attention mechanism that recombines the elements of $\mathcal{B}$ as follows:

$$\boldsymbol{a}_b = \sum_{m=1}^{M} \alpha_{bm} \, \boldsymbol{y}_m^{(k)} , \tag{4.6}$$

where α_{bm} is the attention weight assigned to branch m to compute the context vector of branch b. This mechanism is shown in Fig. 4.2.

As discussed in the previous section, there are many different ways to compute the weights, α_{bm}. We propose computing α_{bm} using a simple one-hidden-layer neural net with the linear activation g, that is:

$$\alpha_{bm} = \frac{\exp(\xi_{bm})}{\sum_{m'} \exp(\xi_{bm'})}\,, \tag{4.7}$$

$$\xi_{bm} = g\left(W_{bm}\boldsymbol{y}_m^{(k)} + \beta_{bm}\right)\,. \tag{4.8}$$

Note that each branch b attends to the representations in $\mathcal{B}$ using the branch-specific parameters $W_{b\cdot}$ and $\beta_{b\cdot}$. Note also that these parameters are not shared across the elements in $\mathcal{B}$, i.e. the model learns different parameters to obtain the logits ξ_{b1}, ξ_{b2}, and ξ_{bB}, which determine how branch b attends to branch m. Of course, it would be possible to use the same parameters for each m, but this reduces the flexibility of the attention module. An interpretation of this approach is as follows: to be linearly combined, the tensors in $\mathcal{B}$ must have a consistent dimension, i.e. we need to fix the dimensionality of the feature maps flowing from the branches to the attention module. One way to compensate for this condition is to introduce a map that projects $\boldsymbol{y}_m^{(k)} \in \mathcal{B}$ into a feature space that disentangles the information needed by branch b about branch m. Our attention module implements that embedding using a fully connected layer with the parameters W_{bm} and β_{bm}.

For further flexibility, we employ residual connections around the attention module; that is, we concatenate $\boldsymbol{a}_b$ with the original latent representation flowing from branch b to the attention module as follows:

$$\tilde{\boldsymbol{a}}_b = \boldsymbol{a}_b \bigoplus \boldsymbol{y}_b^{(k)}\,. \tag{4.9}$$

Once the attention weights have been computed, the predictions for each level of the hierarchy are obtained using a softmax-activated output layer conditioned on $\tilde{\boldsymbol{a}}_b$, i.e.:

$$\boldsymbol{y}_b = \sigma(W_b\tilde{\boldsymbol{a}}_b + \theta_b)\,. \tag{4.10}$$

Note that Equations (4.1) and (4.2), which define the branched approach to hierarchical classification, are still valid. The fundamental difference between our method and the previous techniques is that we place an attention module between the features of (4.2) and the output in (4.3).

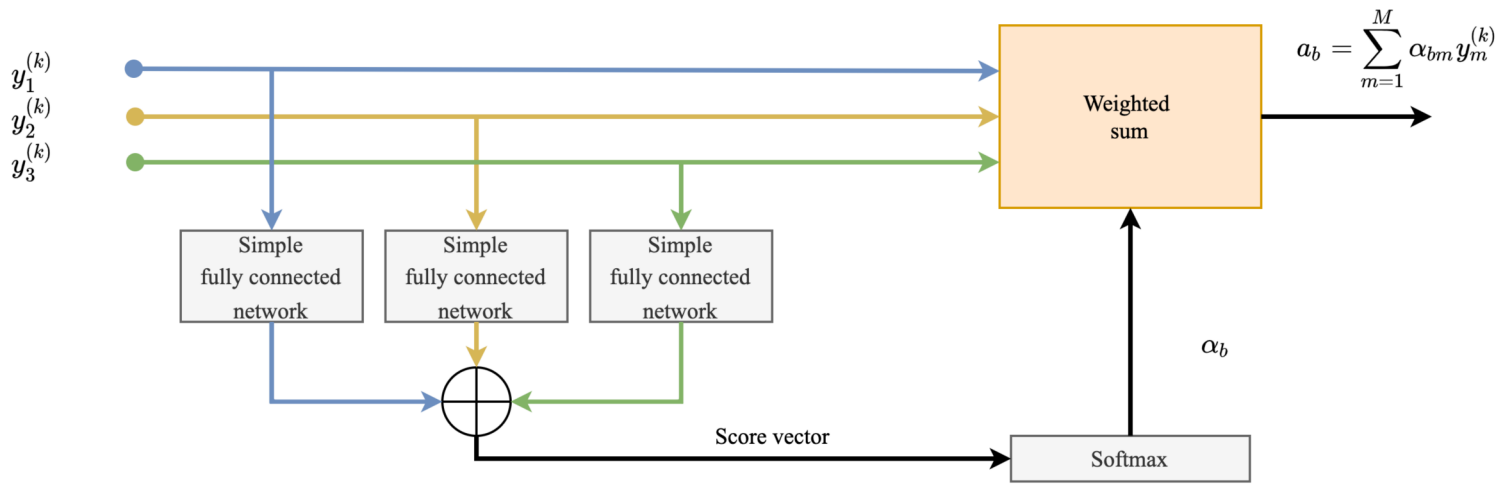


Figure 4.2: Attention mechanism for branch b in a 3-level hierarchy.

4.3 Learning

Provided that all the subnets in the model are differentiable, we can train the system end-to-end using backpropagation. We used the cross-entropy loss to guide the learning of each branch. That is if $\boldsymbol{y}_b^*$ denotes the desired probability distribution for branch b and $\boldsymbol{y}_b = \boldsymbol{y}_b^{(K_b)}$ is the predicted distribution of that branch, the loss corresponding to level b in the taxonomy is computed as:

$$L_b(\boldsymbol{y}^*, \boldsymbol{y}) = -\mathbb{E}_{\boldsymbol{y}^*} \ln(\boldsymbol{y}) . \tag{4.11}$$

In practice, the expected value in (4.11) is estimated using data $S = \{(\boldsymbol{x}^{(n)}, \boldsymbol{y}_b^{(n*)})\}$, which have been annotated for that level of the class hierarchy. Previous works (see, for example, [59] and [58]) often assumed that data were classified at a finer level in the hierarchy. In this case, level-wise annotations can be obtained by tracing the hierarchy back. However, note that the current formulation supports partial annotations.

The loss corresponding to the entire taxonomy is defined as the weighted sum of these losses.

$$L(\boldsymbol{y}_{1:b}^*, \boldsymbol{y}_{1:b}) = \sum_b \omega_b L_b(\boldsymbol{y}_b^*, \boldsymbol{y}_b). \tag{4.12}$$

Chapter 5

Experiments

In this chapter, we describe the experiments conducted to evaluate the proposed method on three image classification datasets: CIFAR-10 [26], CIFAR-100 [26], and Fashion MNIST [55]. These datasets have been widely used in recent studies to assess the performances of deep hierarchical classifiers. We compared the performance of our model with four baselines: B-CNN [59], H-CNN [58], Add-net [19] and Concat-net [19]. We also compared the performance of the hierarchical CNN models with a traditional CNN trained to predict the last level of the taxonomy. We chose B-CNN as a baseline because it is the base architecture on which most recent works are based. On the other hand, H-CNN is the most recent method proposed for this task. In addition, Add-net and Concat-net are recent architectures slightly simpler than HCNN and, thus, are worth considering.

5.1 Datasets

The CIFAR-10 dataset comprises 60,000 32×32 color images organized into ten natural classes, with 6,000 images per class. We employed the three-level class taxonomy proposed by Zhu et al. [59] and then used by [58], which organizes the ten fine-grained classes into seven coarser classes, which are then grouped into two more abstract classes. CIFAR-100 consists of 60,000 32×32 RGB images divided into 100 natural classes with 600 images per class. The class hierarchy used in [59] and [58] organizes the classes into three levels: the first level contains eight coarse categories, the second level has 20 more refined categories, and the third level accommodates the 100 original classes. Fashion MNIST comprises 70,000 28 x 28 gray images extracted from Zalando, an online fashion platform. In [42], the authors built a three-level taxonomy for this dataset which included two classes at the coarsest level: *clothes* and *goods*. The next level separates the class *clothes* into four sub-categories (*tops*, *bottoms*, *dresses*, and *outers*) and the class *goods* into two sub-categories (*accessories* and *shoes*), for a total of six new classes. Finally, the last level accommodates the ten original classes: *t-shirt*, *trouser*, *pullover*, *dress*, *coat*, *sandals*, *shirt*, *sneaker*, *bags*,

and *ankle boots*. We adapted our model and the baselines to learn this taxonomy.

5.2 Performance Metrics

Most current related works evaluate and compare hierarchical classification models using flat measures such as accuracy, precision, and recall. Thus, to measure the level-wise performance, we used the flat metric *accuracy* per level of the hierarchy. However, these measures do not consider the relations between different levels of the hierarchy. Many hierarchical performance metrics have been proposed in the literature [24]. From these, we used the Kiritchenko's metrics [21] because these metrics are suitable for more different hierarchical classification problems, and Silla and Freitas recommended his usage in [44]. In addition, we propose two custom metrics: hierarchical consistency to evaluate the consistency of the predictions and hierarchical accuracy.

5.2.1 Accuracy per level

According to Sammut and Webb [41], accuracy "refers to a measure of the degree to wich the predictions of a model match the reality being modeled". In the context of classification models, accuracy $= P(\lambda(X) = Y)$, where $P(X, Y)$ is a joint distribution of the input data X and the labels Y and the classification model λ is a function $X \to Y$. The accuracy of a classifier is expressed as a value between 0 and 1, and may be calculated as:

$$\text{Accuracy} = \frac{\text{Number of correctly classified objets}}{\text{Total number of objects}}. \tag{5.1}$$

For hierarchical classification, we use this metric for each level of the class hierarchy separately.

5.2.2 Hierarchical Metrics

Hierarchical precision, recall and f-score

To measure the performance of hierarchical classification, Kiritchenko et al. [21] proposed a hierarchical version of the flat metrics precision, recall, and F-score. The authors called the metrics hP (Hierarchical precision), hR (Hierarchical recall), and hF-score (Hierarchical F-score). Formally, let C_i be the true label set for an instance d_i, i.e., the set of elements composed of the ground-truth label for the finest hierarchy level and all its ancestors. On the other hand, let $\hat{C}_i$ be the set of labels predicted for an instance d_i at each level of the class hierarchy. Therefore, hP and hR are computed as follows:

$$hP = \frac{\sum_i |C_i \cap \hat{C}_i|}{\sum_i |\hat{C}_i|}, \qquad hR = \frac{\sum_i |C_i \cap \hat{C}_i|}{\sum_i |C_i|}. \tag{5.2}$$

Then we can combine the two values into one hF-measure:

$$hF_\beta = \frac{(\beta^2 + 1) \cdot hP \cdot hR}{(\beta^2 \cdot hP + hR)}, \beta \in [0, +\inf]. \tag{5.3}$$

In particular, authors recommend $\beta = 1$, giving precision and recall equal weights.

$$hF_1 = \frac{2 \cdot hP \cdot hR}{hP + hR}. \tag{5.4}$$

These metrics fulfilling the following requirements formulated by the authors:

1. The measure gives credit to partially correct classification.
2. For non-mandatory leaf node prediction problems:
   a) The measure gives a higher evaluation for correctly classifying one level down compared to staying at the parent node.
   b) The measure gives a lower evaluation for incorrectly classifying one level down compared to staying at the parent node.
3. The measure punishes errors at higher levels of a hierarchy more heavily.

To exemplify the use of this metric, Table 5.1 shows five instances associated with the taxonomy illustrated in Fig. 5.1. It is worth noticing that if all the classes have labels for every level in the taxonomy (full labeling) and the task is mandatory leaf node prediction (the model always assigns leaf classes), the cardinality of the true label sets will be equal to the cardinality of the predicted labels, so hR, hP, and hF_1-score will have the same numerical value.

Table 5.1: Example of the hierarchical metrics calculation

d_i	True Labels C_i	Predicted Labels $\hat{C}_i$	$\lvert C_i \cap \hat{C}_i \rvert$	$\lvert C_i \rvert$	$\lvert \hat{C}_i \rvert$	$\hat{C}_i \in G$
d_1	{“Cat”, “Toyger”}	{ “Cat”, “Toyger”}	2	2	2	1
d_2	{“Dog”, “Beagle”}	{ “Dog”, “Sphynx”}	1	2	2	0
d_3	{“Cat”, “Sphynx”}	{ “Cat”, “Toyger”}	1	2	2	1
d_4	{“Dog”, “Poodle”, “Toy”}	{ “Dog”, “Toyger”}	1	3	2	0
d_5	{“Dog”, “Poodle”, “Toy”}	{“Dog”, “Beagle”, “Toy”}	2	3	3	0

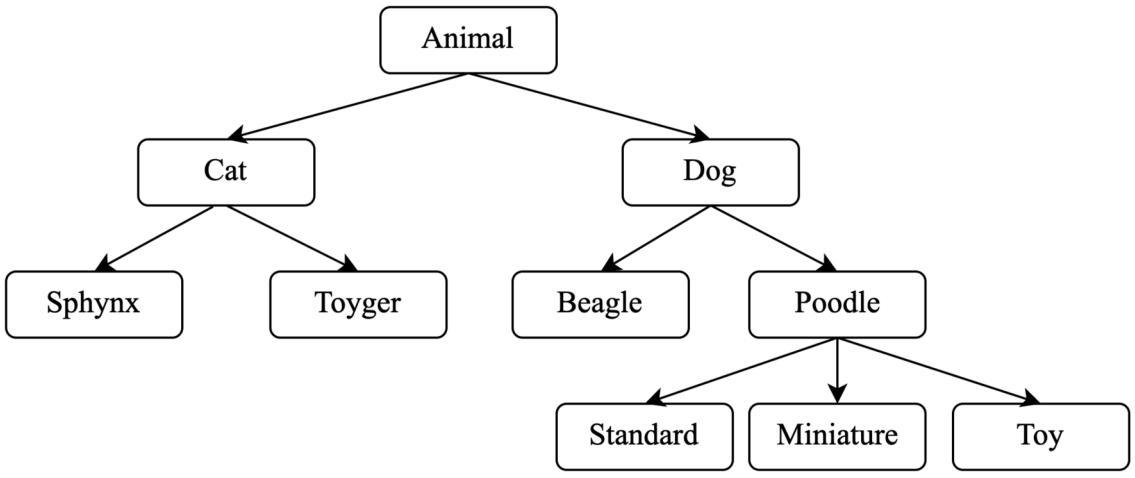


Figure 5.1: Example of a taxonomy to exemplify the use of hierarchical metrics.

Then Kiritchenko's hierarchical metrics are computed as follows:

$$hP = \frac{\sum_i |C_i \cap \hat{C}_i|}{\sum_i |\hat{C}_i|} = \frac{2+1+1+1+2}{2+2+2+2+3} = \frac{7}{11} = 0.64, \tag{5.5}$$

$$hR = \frac{\sum_i |C_i \cap \hat{C}_i|}{\sum_i |C_i|} = \frac{2+1+1+1+2}{2+2+2+3+3} = \frac{7}{12} = 0.58, \tag{5.6}$$

$$hF1 = \frac{2 \cdot hP \cdot hR}{hP + hR} = \frac{2 \cdot 0.64 \cdot 0.58}{0.64 + 0.58} = 0.61. \tag{5.7}$$

Hierarchical Accuracy

In addition to Kiritchenko's metrics, we propose to adapt the standard accuracy metric previously mentioned in Equation 5.1 for hierarchical classification. For this modification, we only consider an object as correctly classified if the entire prediction set for that object is correct. For example, in Table 5.1, we can notice that for five instances, only one is correctily classified (d_1), so hierarchical accuracy is computed as follows:

$$\text{Hierarchical accuracy} = \frac{1}{5} = 0.2. \tag{5.8}$$

Hierarchical Consistency

As argued in [21] and [44], some models are prone to inconsistent class predictions across different levels. This problem is called hierarchical inconsistency or class-membership inconsistency.

Let G be the set of all correct paths from the root to a leaf in the class hierarchy (if the hierarchy is a tree, $|G|$ = the number of class leaves). Then, to verify the hierarchical consistency, we propose to verify that the predicted label set $\hat{C}_i$ for every instance d_i is in G. This metric does not give credit to partially consistent classifications and only considers a prediction as consistent if $\hat{C}_i$ exists in the taxonomy.

It is worth noticing that hierarchical accuracy is always lower than or equal to hierarchical consistency because a prediction can respect the hierarchy but be incorrect for a particular input case (d_3 is an example of this).

For the example presented in Table 5.1, $G = [$“Cat”, “Sphynx”}, {“Cat”, “Toyger”}, {“Dog”, “Beagle”}, {“Dog”, “Poodle”}, {“Dog”, “Poodle”, “Standard”}, {“Dog”, “Poodle”,“Miniature”}, {“Dog”, “Poodle”,“Toy”}]. Then the proposed hierarchical consistency metric (hC) is computed as follows:

$$hC = \frac{\text{No. Consistent Predictions}}{\text{No. Instances}} = \frac{2}{5} = 0.4. \tag{5.9}$$

The previous examples show that while the Kiritchenko's metrics present values close to 0.6, the custom hierarchical consistency and hierarchical accuracy metrics get values of 0.2 and 0.4, respectively. These new metrics will complement the Kiritchenko metric, providing a more severe penalty for errors since they do not penalize entirely correct predictions for the case of hierarchical accuracy and completely correct relationships for the case of hierarchical consistency.

5.3 Experimental Setup

All methods were implemented using fully trainable architectures inspired by VGG models [45], namely, Base-B and Base-C. Both architectures were introduced for hierarchical image classification in [59] and have been employed in many subsequent studies to facilitate comparisons. In all the methods, Base-B and Base-C serve as central feature extractors (central backbone) from which different sub-models or branches predict the different levels of the hierarchy. In [59], authors showed that Base-C outperforms Base-B in all metrics, but we decided to experiment with said central backbone anyway to evaluate its effects.

The four convolutional blocks included in Base-B consist of two convolutional layers with 3 x 3 filters, followed by a 2 x 2 max pooling layer. The convolutional filters are 64, 128, 256, and 512 in the first, second, third, and last blocks, respectively. The activation function was ReLU for all convolutional layers. Inspired by VGG16 [45], Base-C increases the number of convolutional blocks to five and the number of convolutional layers in the last three blocks to three. Furthermore, the fifth block does not have a max pooling layer. Finally, the number of filters was 64, 128, 256, and 512 in the first, second, third, and last two blocks, respectively.

As the taxonomies previously used to validate the baselines have three levels, all the models added three branches to the central backbone. Each branch consisted of two fully-connected layers followed by a Softmax output layer. For models based on the Base-B architecture, the number of neurons for each layer was 256, 512, and 1024 for the first, second, and third branches, respectively. For models based on the Base-C architecture, the number of neurons for each layer was 512, 1024, and 4096 for the first, second, and third branches, respectively. Besides level-specific

branches, H-CNN, Add-net and Concat-net adds skip-connections that connect each branch with the subsequent branches. Following the author's recommendations, we implemented these skip-connections by propagating each branch's input vector to the following branches.

For the comparisons with models using Base-C as the central feature extractor, we chose VGG16 as a flat CNN representative because it is equivalent to B-CNN without the first two branches. On the other hand, for comparisons with models using Base-B, the flat CNN representative is a simplified version of VGG16 used in [59] as a baseline.

As discussed before, all assessed methods use a step-by-step training method named BT-Strategy, which gradually modifies the weights of the multiple losses in the objective function. This strategy encourages the model to focus on coarser hierarchy levels during the first epochs of training and gives more importance to fine-grained predictions at the end of the training. As finding the proper schedule for new datasets is difficult and time-consuming, we evaluated the impact of the BT-Strategy in all experiments.

We conducted two statistical tests to evaluate the significance of the experimental results. First, we employed Friedman's test to assess the (null) hypothesis that the methods we compared were statistically equivalent under a given metric. In this design, the method serves as the group variable, and the level of supervision serves as the blocking variable. Second, when rejecting the null hypothesis of Friedman's test, we compared the proposed method with the other algorithms using the Wilcoxon test with Bonferroni correction to check for pairwise differences. Note that the Bonferroni correction yields much more conservative p-values than those obtained by assuming independence of the pairwise differences. For all the tables, the highlighted values correspond to the maximum per category as long as it has a statistically significant difference at 5% level from the value that follows it. If there is more than one highlighted value, it is because the statistical tests show them as equivalent, showing both as the best value. All the models were implemented using Python with the functional Deep Learning API Keras, running on a desktop computer detailed in Appendix A.

The specific experimental settings for each dataset are the following:

CIFAR-10

For this dataset, we trained all the models using the settings of [59] and [58]. Stochastic Gradient Descent (SGD) was iterated for 60 epochs using a batch size of 128. The learning rate schedule started with a value of 0.003, decreased to 0.0005 after 42 epochs, and to 0.0001 after 52 epochs. For the BT-Strategy, we start with weights of 0.98, 0.01, and 0.01 for the first, second, and third hierarchy levels, respectively. These weights were modified to 0.1, 0.8, and 0.1 after 10 epochs; to 0.1, 0.2, and 0.7 after 20 epochs; and 0, 0, and 1 after 30 epochs. For the experiments without the BT-Strategy, we set the weights to 0.33, 0.33, and 0.34 during the entire training

process.

CIFAR-100

For CIFAR-100, all the models used the Base-C architecture as the main building block. The branches used to predict the different levels of the hierarchy were organized following [59]. We trained all the models for 80 epochs using SGD. The learning rate starts with a value of 0.001. At epoch 55, it decreases to 0.0002. Finally, at epoch 70, it is updated to 0.00005. The schedule of weights for the BT-Strategy was the same as for CIFAR-10.

Fashion MNIST

For all the experiments, we adopted the Base-C architecture detailed in the previous sections and the training settings described in [42] for Fashion MNIST. We trained all the models using vanilla SGD for 60 epochs. The learning rate starts with a value of 0.001. It decreases to 0.0002 after 42 epochs and to 0.00005 after 52 epochs. When the BT-Strategy was active, the weights for the first, second, and third hierarchy levels were 0.98, 0.01, and 0.01, respectively. These weights were modified to 0.1, 0.8, and 0.1 after 15 epochs; to 0.1, 0.2, and 0.7 after 25 epochs; and 0, 0, and 1 after 35 epochs.

5.4 Hyperparameter Tuning

As we mentioned in Chapter 4, we can adapt our proposed method to any branched architecture. The parameters to consider for tuning were the dimensionality of the branches and the number of neurons of the attention module's hidden layers. We performed an exhaustive grid search for all models using Base-C as the central backbone. However, we do not consider Base-B for this grid search because in [59], authors showed that it consistently underperforms Base-C.

To tune our proposed model for CIFAR-10, we considered the following parameters:

- Branch No. Neurons: 16, 32, 64, 128, 256, 512, 1024, 4096.
- Attention module's No. Neurons: 32, 64, 128, 256, 512, 2048.
- Use of BT-Strategy: True or False.

The above resulted in 96 combinations. For all cases, we calculated all the performance metrics using 5-fold cross-validation. Fig. 5.2 presents the 5-fold hierarchical accuracy metric for all combinations. We can see that the optimal setting corresponds to choosing a branch dimensionality of 32, 2048 neurons for the attention module's hidden layers, and BT-Strategy True.

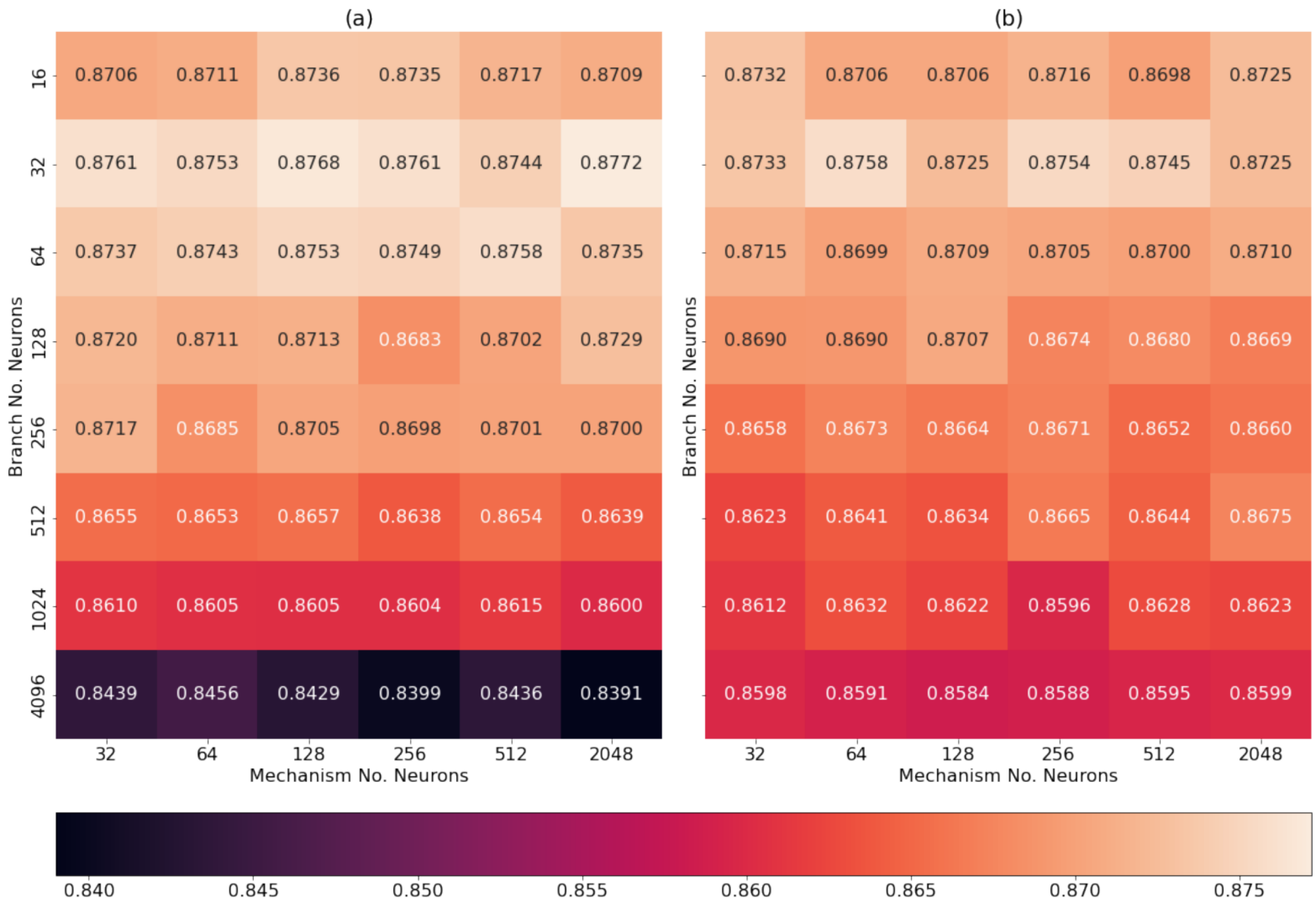


Figure 5.2: 5-fold Hierarchical accuracy results on CIFAR-10. (a) with BT-Strategy, (b) without BT-Strategy.

As shown in Fig. 5.3, increasing the number of neurons of the attention modules for a fixed dimension of the branches produces an increase in the hierarchical accuracy metric, reaching the optimal value at 32 in most cases. However, after reaching the optimum, performance starts to drop, more so when using the BT-Strategy than when it is not. Furthermore, we can observe that the model is very robust to the choice of the attention mechanism's number of neurons with a general trend to (slightly) increase as we increase the dimensionality. The branches' dimensionality is the main factor to determine the model's proneness to overfitting. Generally, small values of this parameter obtains the best results. Furthermore, the breaking point at the left (underfitting) is significantly more difficult to reach than the breaking point at the right (overfitting). The above suggests that the proposed attention module does not need a large number of trainable parameters to work effectively. We attribute this result to the module's flexibility. The model can dynamically select the pieces of data that are useful to make a prediction, and thus it does not need to explicitly keep track of many individual patterns.

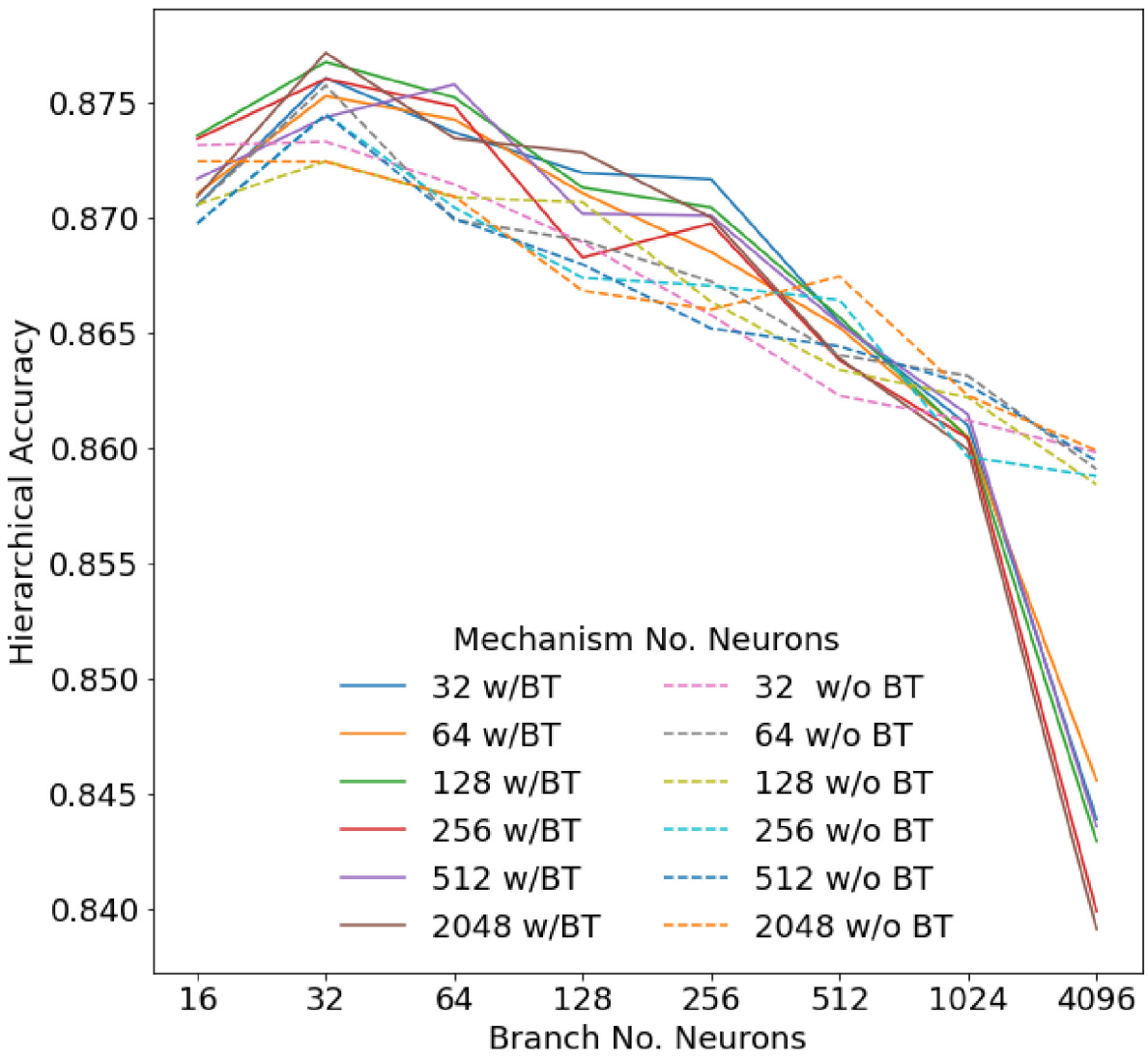


Figure 5.3: Hierarchical accuracy on CIFAR-10 as we increase the number of neurons in the branches.

For CIFAR-100, we considered the following parameters:

- Branch No. Neurons: 64, 128, 256, 512, 1024, 4096.
- Attention module's No. Neurons: 32, 64, 128, 256, 512, 2048.
- Use of BT-Strategy: True or False.

The above resulted in 72 combinations. For all cases, we calculated all the performance metrics using 5-fold cross-validation. Fig. 5.4 presents the 5-fold hierarchical accuracy metric for all combinations. Results shows that choosing a branch dimensionality of 256 and 2048 neurons for the attention module's hidden layers is the optimal setting. Contrary to CIFAR-10 results, the exhaustive grid search suggested not using BT-Strategy. We attribute the previous result to the fact that for this dataset, which has a more significant number of classes, the BT-Strategy takes less importance than the guidance provided by the attention modules.

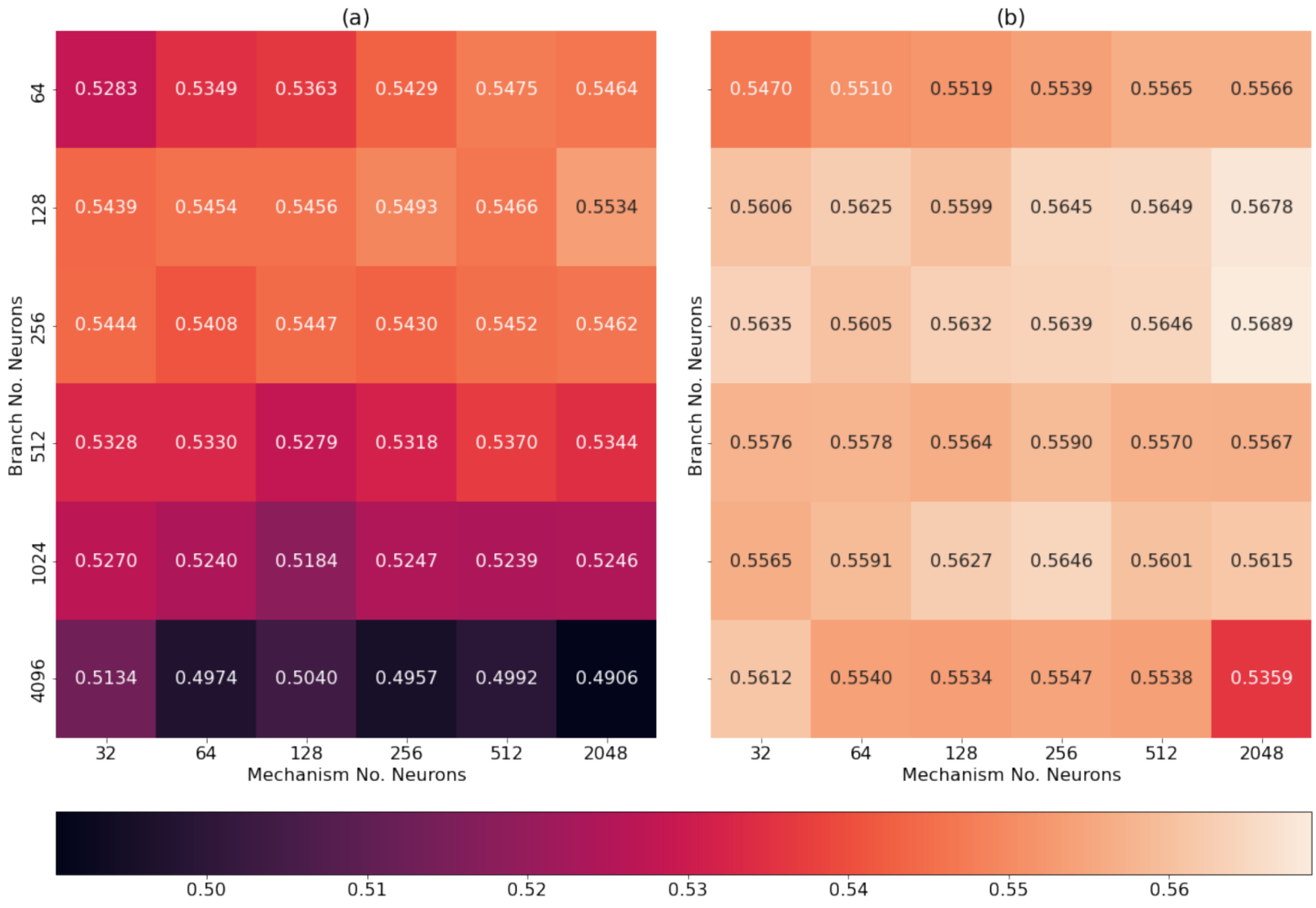


Figure 5.4: 5-fold Hierarchical accuracy results on CIFAR-100. (a) with BT-Strategy, (b) without BT-Strategy.

As shown in Fig. 5.5, when increasing the number of neurons of the attentional modules for a fixed dimension of the branches, the model behaves similarly to CIFAR-10, but with the difference that in most cases, not using the BT-Strategy results in better performance.

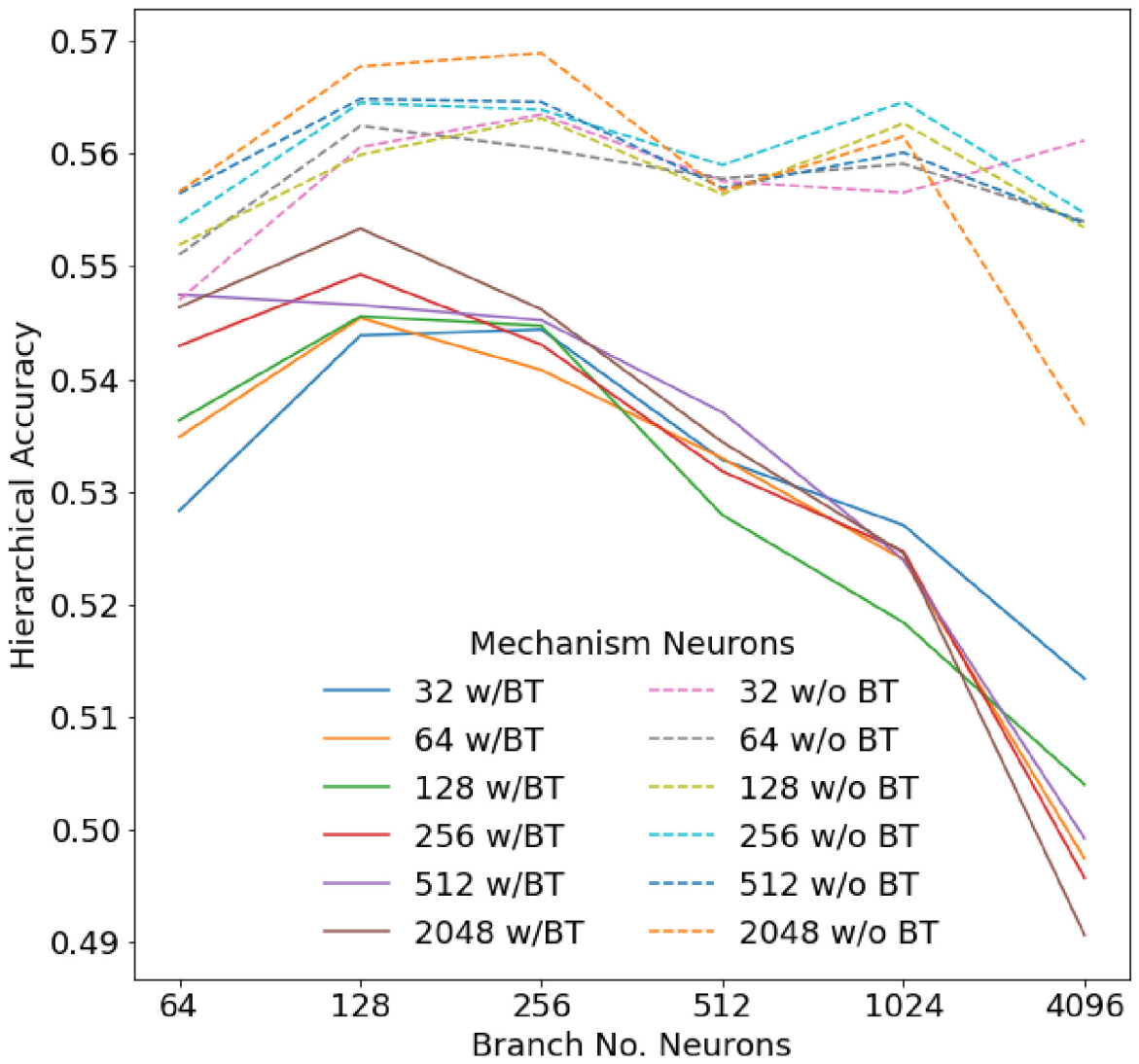


Figure 5.5: Hierarchical accuracy on CIFAR-100 as we increase the number of neurons in the branches.

For Fashion MNIST, we considered the following parameters:

- Branch No. Neurons: 8, 16, 32, 64, 128, 256, 512, 1024, 4096.
- Attention module's No. Neurons: 32, 64, 128, 256, 512, 2048.
- Use of BT-Strategy: True or False.

The above resulted in 72 combinations. The same as for the previous cases, we calculated all the performance metrics using 5-fold cross-validation.

Fig. 5.6 presents the 5-fold hierarchical accuracy metric for all the combinations. We can see that the optimal setting corresponds to choosing a branch dimensionality of 16, 2048 neurons for the attention module's hidden layers and using BT-Strategy.

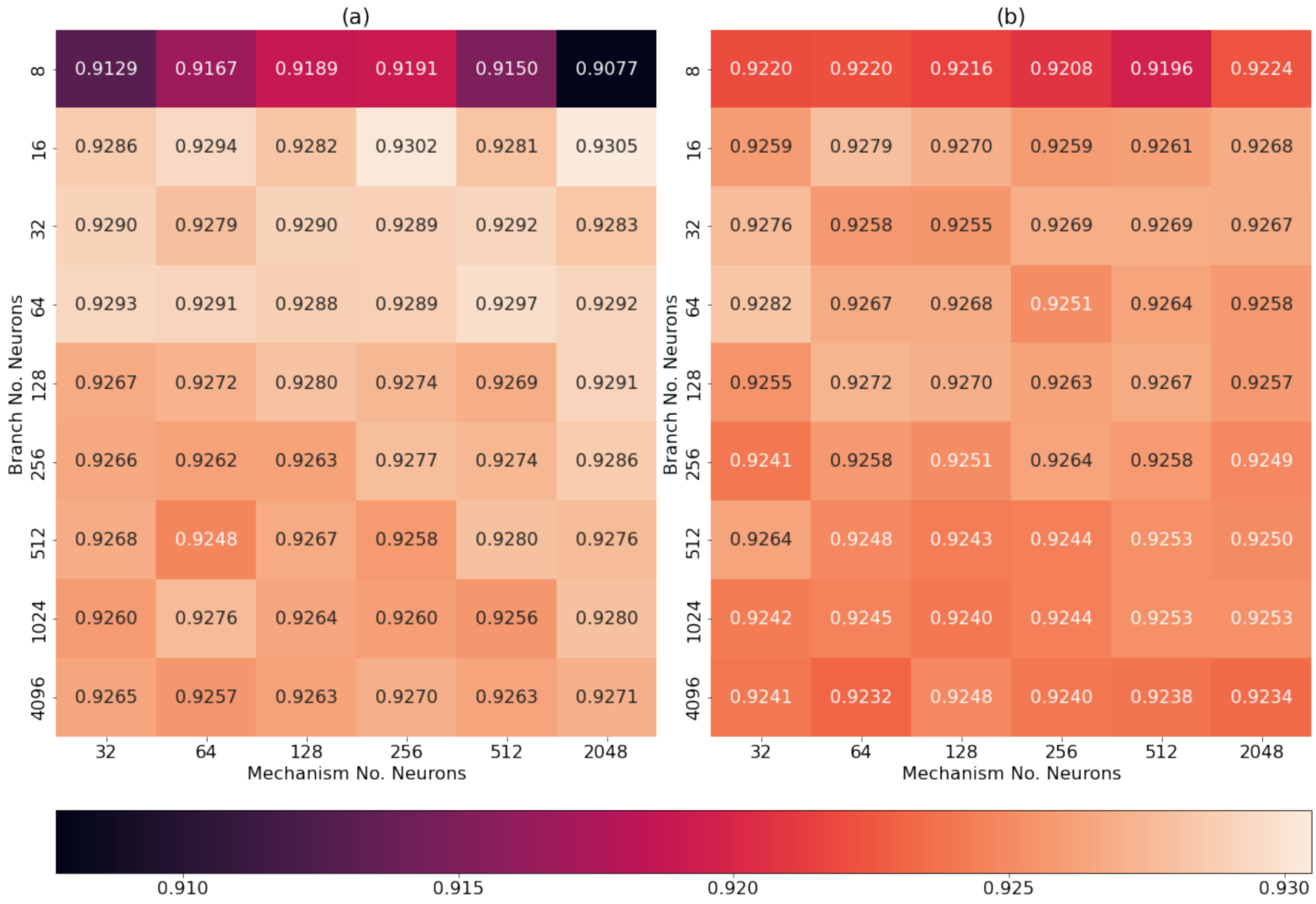


Figure 5.6: 5-fold Hierarchical accuracy results on Fashion MNIST. (a) with BT-Strategy, (b) without BT-Strategy.

As shown in Fig. 5.7, when increasing the number of neurons of the attentional modules for a fixed dimension of the branches, the model's performance exhibits only a slight decline, except for the case with a value of eight neurons, where the model declines sharply.

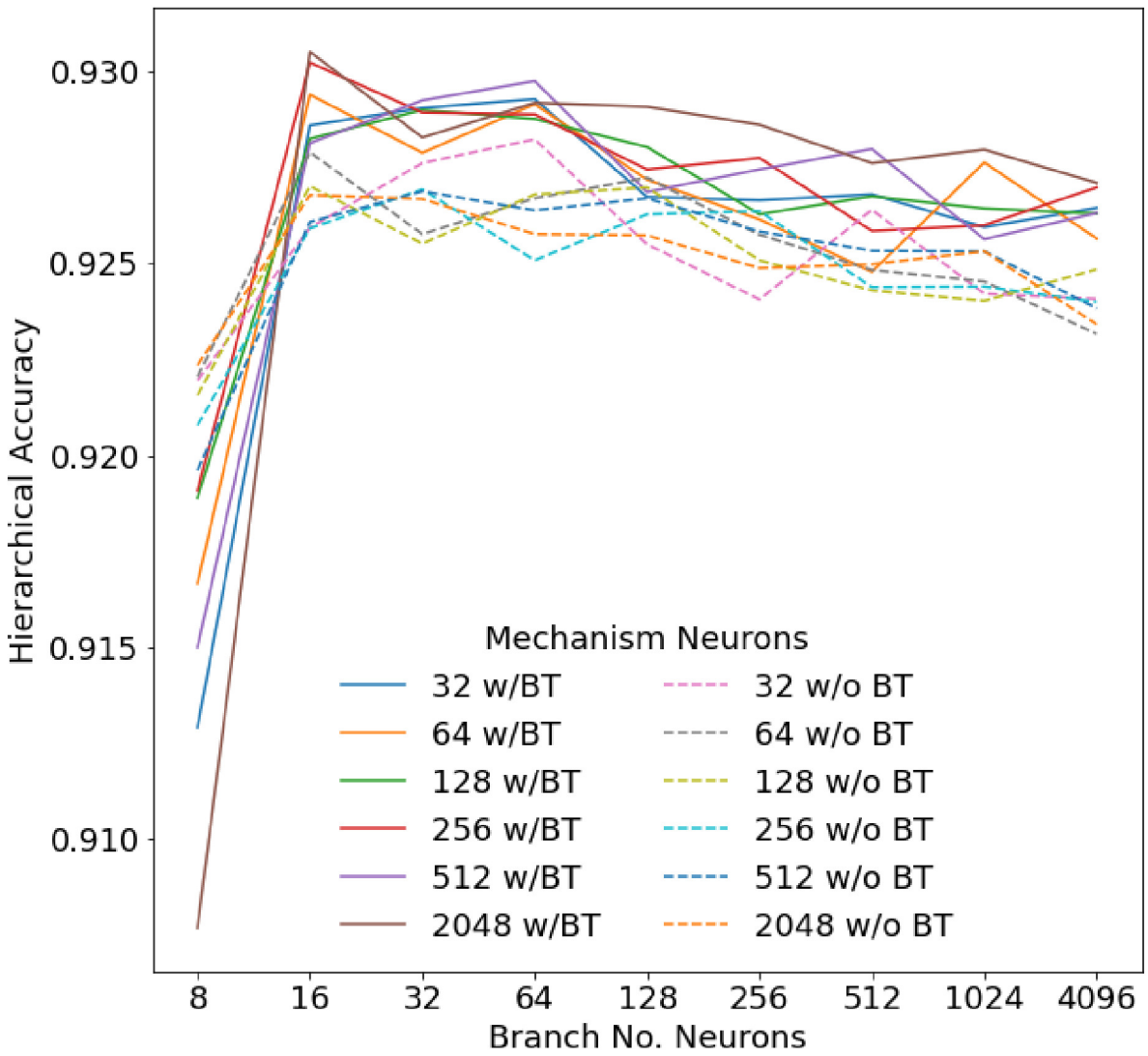


Figure 5.7: Hierarchical accuracy on Fashion MNIST as we increase the number of neurons in the branches.

Finally, Table. 5.2 presents the selected parameters of the best models for the three datasets.

Table 5.2: Best hyperparameters after the exhaustive grid search for CIFAR-10, CIFAR-100 and Fashion MNIST

Dataset	Branch No. Neurons	Mechanism No. Neurons	BT-Strategy
CIFAR-10	32	2048	True
CIFAR-100	256	2048	False
Fashion MNIST	16	2048	True

5.5 Experiment 1: Comparison using author's proposed models

For this experiment, we compared the performance of the proposed model against the baselines using the author's proposed architectures, adapted to the three datasets.

For our attention modules, we implemented fully-connected nets with one hidden layer of 64 neurons each. As the method linearly combines the feature vectors extracted from each branch, these vectors must have a consistent dimension. Therefore, we fixed the dimensionality of the hidden layers in all three branches in our model to 256. This decision was a parsimonious choice that reduced the total number of parameters in the model. Subsequently, we studied the effect of this decision. Since Add-net also linearly combines feature vectors from each branch, we adopted the same standard for this model.

5.5.1 CIFAR-10

Table 5.3 and 5.4 presents the hierarchical and level-wise performance metrics for the different models trained with and without BT-Strategy. The best results within the level of statistical significance are in bold. For details about the p-values of the statistical tests performed, please refer to Tables B.1 and B.2 from the Appendix B.

Table 5.3 shows that our method (BA-CNN) outperformed all the baselines regarding hierarchical accuracy and consistency with statistically significant differences. This result confirms that our attention mechanism can equalize the information flow among the branches, enabling the model to make more accurate predictions respecting the labels' hierarchy. Furthermore, this result is independent of the chosen central backbone (Base-B or Base-C) nor the usage of BT-Strategy, which avoids a cumbersome calibration procedure in new datasets. Regarding hierarchical F1, the proposed model outperformed the baselines in all situations, only excluding using Base-B without BT-Strategy, where even that obtains the best numerical value shares first place with H-CNN and Concat-net.

Regarding level-wise performance, we generally found significant differences at the coarser levels of the hierarchy, in which our method performed better than the other approaches. This result is not surprising because the connectivity patterns of the baselines are focused on fine-grained classifications, implicitly disregarding their consistency with coarser levels.

In this experiment, we also confirmed that Base-C outperformed Base-B models, like in [59], so we only considered Base-C in subsequent experiments.

Table 5.3: Main results on CIFAR-10 (hierarchical metrics)

	Hierarchical Metrics			Param.
Model Base-B	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.7871 (2.57e-03)	0.9006 (1.99e-03)	0.8895 (1.54e-03)	12.38
BA-CNN	**0.8148** (3.43e-03)	**0.9495** (1.84e-03)	**0.8958** (1.81e-03)	8.72
H-CNN [58]	0.7876 (1.91e-03)	0.9030 (3.18e-03)	0.8892 (1.12e-03)	29.160
Add-net [19]	0.7946 (2.07e-03)	0.9309 (2.62e-03)	0.8859 (1.15e-03)	8.57
Concat-net [19]	0.7926 (2.49e-03)	0.9226 (1.73e-03)	0.8864 (1.19e-03)	12.39
Model Base-C				
B-CNN [59]	0.8255 (1.52e-03)	0.8997 (1.48e-03)	0.9168 (8.70e-04)	49.67
BA-CNN	**0.8751** (3.33e-03)	**0.9803** (2.31e-03)	**0.9284** (1.77e-03)	18.75
H-CNN [58]	0.8249 (1.77e-03)	0.9070 (2.80e-03)	0.9144 (7.57e-04)	108.39
Add-net [19]	0.8332 (1.23e-03)	0.9212 (1.53e-03)	0.9138 (7.84e-04)	18.60
Concat-net [19]	0.8312 (2.26e-03)	0.9212 (2.85e-03)	0.9127 (8.38e-04)	49.69
Without BT-Strategy				
Model Base-B				
B-CNN [59]	0.7774 (4.09e-03)	0.9051 (2.36e-03)	0.8842 (2.33e-03)	12.38
BA-CNN	**0.8103** (2.88e-03)	**0.9677** (2.46e-03)	**0.8891** (1.79e-03)	8.72
H-CNN [58]	0.7867 (2.01e-03)	0.9138 (2.08e-03)	**0.8875** (9.03e-04)	29.160
Add-net [19]	0.7928 (2.76e-03)	0.9376 (1.68e-03)	0.8855 (1.61e-03)	8.57
Concat-net [19]	0.7881 (2.62e-03)	0.9280 (1.93e-03)	**0.8850** (1.81e-03)	12.39
Model Base-C				
B-CNN [59]	0.8217 (2.17e-03)	0.9023 (2.23e-03)	0.9152 (7.66e-04)	49.67
BA-CNN	**0.8724** (1.56e-03)	**0.9859** (1.17e-03)	**0.9253** (1.11e-03)	18.75
H-CNN [58]	0.8215 (1.85e-03)	0.9049 (1.64e-03)	0.9131 (5.75e-04)	108.39
Add-net [19]	0.8305 (1.10e-03)	0.9174 (2.10e-03)	0.9160 (4.93e-04)	18.60
Concat-net [19]	0.8263 (1.79e-03)	0.9133 (2.02e-03)	0.9150 (7.85e-04)	49.69

Table 5.4: Main results on CIFAR-10 (flat metrics)

	Accuracy by Level			Param.
Model Base-B	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.8267 (2.64e-03)	7.86
B-CNN [59]	0.9589 (8.75e-04)	0.8687 (2.33e-03)	**0.8409** (3.02e-03)	12.38
BA-CNN	**0.9683** (1.19e-03)	**0.8796** (2.21e-03)	**0.8395** (2.64e-03)	8.72
H-CNN [58]	0.9585 (1.37e-03)	0.8702 (1.45e-03)	**0.8390** (1.62e-03)	29.16
Add-net [19]	0.9577 (1.87e-03)	0.8658 (1.53e-03)	0.8342 (2.08e-03)	8.57
Concat-net [19]	0.9570 (1.49e-03)	0.8662 (1.70e-03)	0.8360 (2.52e-03)	12.39
Model Base-C				
Flat CNN [45]	-	-	0.8823 (8.53e-04)	39.98
B-CNN [59]	0.9592 (1.34e-03)	0.9049 (1.51e-03)	**0.8862** (2.06e-03)	49.67
BA-CNN	**0.9823** (6.70e-04)	**0.9191** (2.88e-03)	**0.8837** (2.48e-03)	18.76
H-CNN [58]	0.9604 (1.46e-03)	0.9019 (1.47e-03)	0.8809 (1.17e-03)	108.39
Add-net [19]	0.9589 (1.08e-03)	0.8994 (1.63e-03)	0.8831 (1.46e-03)	18.60
Concat-net [19]	0.9578 (1.39e-03)	0.8994 (1.43e-03)	0.8810 (1.69e-03)	49.69
Without BT-Strategy				
Model Base-B				
B-CNN [59]	0.9640 (1.34e-03)	0.8665 (3.76e-03)	0.8222 (3.86e-03)	12.38
BA-CNN	**0.9724** (8.54e-04)	**0.8717** (2.45e-03)	0.8233 (2.86e-03)	8.72
H-CNN [58]	0.9653 (1.45e-03)	**0.8713** (2.01e-03)	0.8260 (2.27e-03)	29.16
Add-net [19]	0.9654 (9.38e-04)	**0.8675** (1.48e-03)	0.8236 (3.18e-03)	8.57
Concat-net [19]	0.9645 (1.42e-03)	**0.8667** (2.30e-03)	0.8238 (2.68e-03)	12.39
Model Base-C				
B-CNN [59]	0.9622 (8.99e-04)	0.9094 (1.07e-03)	0.8742 (1.58e-03)	49.67
BA-CNN	**0.9824** (3.61e-04)	**0.9161** (1.72e-03)	0.8775 (1.84e-03)	18.75
H-CNN [58]	0.9617 (1.19e-03)	0.9035 (1.12e-03)	0.8743 (1.67e-03)	108.39
Add-net [19]	0.9643 (1.07e-03)	0.9067 (1.30e-03)	0.8770 (8.79e-04)	18.60
Concat-net [19]	0.9621 (8.16e-04)	0.9079 (1.57e-03)	0.8748 (1.08e-03)	49.69

5.5.2 CIFAR-100

Table 5.5 and 5.6 present hierarchical and level-wise performance metrics for CIFAR-100. The best results within the level of statistical significance are in bold. For details about the p-values of the statistical tests performed, please refer to Tables B.4 and B.5 from the Appendix B.

For this dataset, we can see that our method achieved the highest values for all the hierarchical metrics. In particular, hierarchical accuracy was at least ten points higher than baseline values, and hierarchical consistency was higher with margins of at least 25 points. The above demonstrates more clearly the principal advantages of the proposed model in hierarchical terms. We obtained this result despite having only 40% of the number of parameters used by B-CNN and 18% of the number of parameters required by H-CNN to implement its highly dense connectivity pattern. Furthermore, all of these findings are independent of the BT-Strategy, which, as previously discussed, could bias the models towards fine-grained class levels at the expense of shallower ones.

Regarding level-wise performance, as in the previous experiment, our method outperformed the baselines at the first two levels of the class hierarchy. Furthermore, in some cases, this advantage was noticeably greater than the slight decrease observed at the third level. For instance, using BT-Strategy, we improve the baselines' accuracy at the coarser level by more than 10 points, whereas the decrease at the third level is 4 points in the worst case. Furthermore, all of these findings are independent of the BT-Strategy, which, as previously discussed, could bias the models towards fine-grained class levels at the expense of shallower ones.

Table 5.5: Main results on CIFAR-100 (hierarchical metrics)

	Hierarchical Metrics			Param.
Model	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.4478 (2.22e-03)	0.5842 (3.03e-03)	0.6877 (1.36e-03)	50.06
BA-CNN	**0.5721** (3.59e-03)	**0.8540** (3.47e-03)	**0.7286** (2.27e-03)	18.81
H-CNN [58]	0.4639 (2.81e-03)	0.6092 (3.10e-03)	0.6932 (1.18e-03)	108.78
Add-net [19]	0.4285 (3.79e-03)	0.5722 (3.74e-03)	0.6596 (1.95e-03)	18.63
Concat-net [19]	0.4510 (2.66e-03)	0.5971 (4.70e-03)	0.6751 (1.27e-03)	50.22
Without BT-Strategy				
B-CNN [59]	0.4638 (2.76e-03)	0.6050 (3.91e-03)	0.6976 (2.00e-03)	50.06
BA-CNN	**0.5854** (3.94e-03)	**0.9028** (2.44e-03)	**0.7307** (3.43e-03)	18.81
H-CNN [58]	0.4773 (2.18e-03)	0.6245 (3.31e-03)	0.7005 (6.20e-04)	108.78
Add-net [19]	0.4771 (3.09e-03)	0.6510 (3.45e-03)	0.6911 (1.12e-03)	18.63
Concat-net [19]	0.4723 (3.11e-03)	0.6272 (4.92e-03)	0.6920 (1.21e-03)	50.22

Table 5.6: Main results on CIFAR-100 (flat metrics)

	Accuracy by Level			Param.
Model	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.6256 (2.48e-03)	40.35
B-CNN [59]	0.7230 (1.79e-03)	0.7021 (2.06e-03)	0.6380 (3.36e-03)	50.06
BA-CNN	**0.8333** (2.06e-03)	**0.7380** (3.25e-03)	0.6147 (3.16e-03)	18.81
H-CNN [58]	0.7279 (2.12e-03)	0.7014 (2.21e-03)	**0.6504** (2.76e-03)	108.78
Add-net [19]	0.6961 (4.32e-03)	0.6480 (2.88e-03)	0.6345 (2.62e-03)	18.63
Concat-net [19]	0.7102 (3.10e-03)	0.6793 (2.36e-03)	0.6358 (2.30e-03)	50.22
Without BT-Strategy				
B-CNN [59]	0.7413 (2.62e-03)	0.7320 (2.28e-03)	0.6194 (2.85e-03)	50.06
BA-CNN	**0.8415** (4.00e-03)	**0.7427** (3.63e-03)	0.6080 (3.68e-03)	18.81
H-CNN [58]	0.7409 (1.99e-03)	0.7175 (2.69e-03)	**0.6432** (2.67e-03)	108.78
Add-net [19]	0.7430 (2.93e-03)	0.7096 (3.96e-03)	0.6208 (2.64e-03)	18.63
Concat-net [19]	0.7391 (2.69e-03)	0.7157 (2.07e-03)	0.6213 (1.77e-03)	50.22

5.5.3 Fashion MNIST

Table 5.7 and 5.8 summarize the results of the different models on Fashion MNIST. The best results within the level of statistical significance are in bold. For details about the p-values of the statistical tests performed, please refer to Tables B.7 and B.8 from the Appendix B.

In this dataset, our method outperforms all others by statistically significant margins regarding hierarchical accuracy and consistency. The above is a recurrent result across all the datasets that confirms the model is learning the class taxonomy better. Regarding accuracy per level, all the methods achieve similar results with remarkably narrow numerical differences that the statistical tests do not find significant in the vast majority of cases. We must note that in this dataset, the B-CNN model already obtains an excellent performance at the coarsest level of the hierarchy (about 0.998 of accuracy), significantly reducing the margin of improvement that its variants can achieve, including the ours, which usually improves results at that level.

The above results suggest that the proposed method is a reliable extension of the referenced model: it can improve B-CNN in complex tasks, keep its performance in more straightforward tasks, and improve hierarchical accuracy and consistency.

Table 5.7: Main results on Fashion MNIST (hierarchical metrics)

	Hierarchical Metrics			Param.
Model	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.9206 (1.36e-03)	0.9807 (1.03e-03)	0.9644 (6.39e-04)	40.56
BA-CNN	**0.9262** (2.22e-03)	**0.9903** (1.00e-03)	**0.9650** (1.05e-03)	17.41
H-CNN [58]	0.9232 (1.49e-03)	0.9794 (1.25e-03)	**0.9659** (5.85e-04)	82.11
Add-net [19]	0.9218 (1.78e-03)	0.9814 (1.26e-03)	**0.9648** (7.14e-04)	17.25
Concat-net [19]	0.9209 (1.68e-03)	0.9809 (1.34e-03)	0.9644 (6.91e-04)	40.58
Without BT-Strategy				
B-CNN [59]	0.9177 (2.30e-03)	0.9823 (1.42e-03)	0.9629 (9.34e-04)	40.56
BA-CNN	**0.9262** (1.33e-03)	**0.9942** (1.04e-03)	0.9642 (6.78e-04)	17.41
H-CNN [58]	0.9203 (1.84e-03)	0.9810 (1.24e-03)	0.9644 (8.45e-04)	82.11
Add-net [19]	0.9189 (1.30e-03)	0.9829 (9.16e-04)	0.9632 (6.20e-04)	17.25
Concat-net [19]	0.9192 (1.14e-03)	0.9832 (6.37e-04)	0.9632 (5.52e-04)	40.58

Table 5.8: Main results on Fashion MNIST (flat metrics)

	Accuracy by Level			Param.
Model	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.9299 (1.50e-03)	33.69
B-CNN [59]	0.9978 (3.25e-04)	**0.9645** (1.25e-03)	**0.9309** (1.17e-03)	40.56
BA-CNN	0.9977 (2.97e-04)	**0.9662** (1.30e-03)	**0.9312** (2.25e-03)	17.41
H-CNN [58]	0.9981 (2.79e-04)	**0.9657** (1.06e-03)	**0.9339** (1.52e-03)	82.11
Add-net [19]	0.9980 (2.61e-04)	**0.9645** (1.05e-03)	**0.9318** (1.42e-03)	17.25
Concat-net [19]	0.9979 (2.05e-04)	0.9638 (1.14e-03)	**0.9314** (1.43e-03)	40.58
Without BT-Strategy				
B-CNN [59]	0.9979 (1.78e-04)	**0.9648** (1.00e-03)	0.9260 (2.07e-03)	40.56
BA-CNN	0.9980 (2.16e-04)	**0.9660** (1.14e-03)	**0.9286** (1.22e-03)	17.41
H-CNN [58]	0.9978 (2.54e-04)	**0.9666** (9.61e-04)	**0.9288** (1.86e-03)	82.11
Add-net [19]	0.9978 (2.47e-04)	0.9645 (1.25e-03)	0.9273 (1.44e-03)	17.25
Concat-net [19]	0.9978 (3.67e-04)	0.9647 (7.66e-04)	0.9271 (1.36e-03)	40.58

5.6 Experiment 2: Comparison using hypertuned models

Here, we compare our model with the other methods using the hyperparameters presented in Section 5.4 with the baselines. We also adjusted the baselines to that hyperparameters to make a fair comparison and evaluate the complexity of the models in the same scenario.

As a first general observation, we can see from Table 5.10, 5.9, 5.12, 5.11, 5.14, and 5.13 that, as expected, the version of the model using the best hyperparameters outperformed the parsimonious model considered in the previous experiment in all the metrics for CIFAR-10 and Fashion MNIST. For CIFAR-100, the difference between the tuned and parsimonious models is not statistically significant, so we presented the latter as a convenient choice that reduces the number of parameters.

Second, the model selection process showed that our model requires a lower dimensionality of the branches compared to the architecture used in Section 5.5. When adapting the flat CNN baseline to that setting, this model improved its performance at the third hierarchy level in all the datasets. As all models use the flat CNN as their central backbone, most models also improved their performance at the third hierarchy level.

Finally, the conclusions presented in Section 5.5 remain valid, and hyper-parameter tuning only improves the differences. The most affected model is H-CNN, which suffers a decrease of almost 3% in the accuracy on level three for CIFAR-100.

Table 5.9: Results for hierarchical metrics on CIFAR-10

	Hierarchical Metrics			Param.
Model	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.8265 (2.39e-03)	0.8996 (1.62e-03)	0.9162 (9.51e-04)	15.186
BA-CNN	**0.8829** (1.52e-03)	**0.9877** (1.59e-03)	**0.9315** (9.43e-04)	15.813
H-CNN [58]	0.8276 (1.40e-03)	0.9027 (2.26e-03)	0.9160 (9.04e-04)	15.842
Add-net [19]	0.8405 (2.64e-03)	0.9272 (2.47e-03)	0.9171 (1.59e-03)	15.186
Concat-net [19]	0.8389 (2.55e-03)	0.9324 (2.31e-03)	0.9141 (1.28e-03)	15.187

Table 5.10: Results for flat metrics on CIFAR-10

	Accuracy by Level			Param.
Model	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.8904 (1.69e-03)	14.765
B-CNN [59]	0.9562 (6.74e-04)	0.9014 (1.77e-03)	**0.8909** (2.25e-03)	15.186
BA-CNN	**0.9836** (6.93e-04)	**0.9230** (1.16e-03)	**0.8880** (1.65e-03)	15.813
H-CNN [58]	0.9570 (6.34e-04)	0.9010 (1.77e-03)	**0.8900** (1.97e-03)	15.842
Add-net [19]	0.9599 (1.91e-03)	0.9038 (2.42e-03)	0.8875 (1.54e-03)	15.186
Concat-net [19]	0.9601 (1.63e-03)	0.8983 (1.35e-03)	0.8839 (1.90e-03)	15.187

Table 5.11: Results for hierarchical metrics on CIFAR-100

	Hierarchical Metrics			Param.
Model	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.4639 (2.19e-03)	0.6116 (3.13e-03)	0.6971 (7.11e-04)	18.627
BA-CNN	**0.5887** (3.26e-03)	**0.9025** (4.10e-03)	**0.7327** (2.21e-03)	23.415
H-CNN [58]	0.4641 (2.14e-03)	0.6123 (2.87e-03)	0.6934 (1.32e-03)	23.870
Add-net [19]	0.4765 (2.70e-03)	0.6506 (3.41e-03)	0.6908 (1.51e-03)	18.627
Concat-net [19]	0.4796 (3.88e-03)	0.6582 (4.04e-03)	0.6917 (1.39e-03)	18.685

Table 5.12: Results for flat metrics on CIFAR-100

	Accuracy by Level			Param.
Model	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.6292 (2.15e-03)	15.35
B-CNN [59]	0.7418 (2.60e-03)	0.7251 (2.19e-03)	**0.6243** (3.20e-03)	18.627
BA-CNN	**0.8409** (1.49e-03)	**0.7447** (2.66e-03)	0.6124 (3.63e-03)	23.415
H-CNN [58]	0.7444 (1.87e-03)	0.7193 (1.93e-03)	0.6163 (3.36e-03)	23.870
Add-net [19]	0.7431 (3.01e-03)	0.7091 (3.91e-03)	**0.6203** (3.51e-03)	18.627
Concat-net [19]	0.7412 (2.28e-03)	0.7097 (1.85e-03)	**0.6242** (3.99e-03)	18.685

Table 5.13: Results for hierarchical metrics on Fashion MNIST

	Hierarchical Metrics			Param.
Model	Accuracy	Consistency	F1	(MM)
B-CNN [59]	0.9187 (1.65e-03)	0.9777 (1.28e-03)	**0.9636** (8.20e-04)	14.869
BA-CNN	**0.9280** (2.21e-03)	**0.9932** (7.91e-04)	**0.9652** (1.25e-03)	15.201
H-CNN [58]	0.9174 (1.46e-03)	0.9767 (2.15e-03)	0.9633 (6.65e-04)	15.106
Add-net [19]	0.9208 (3.85e-03)	0.9823 (3.48e-03)	**0.9641** (1.29e-03)	14.869
Concat-net [19]	**0.9248** (1.33e-03)	0.9860 (1.56e-03)	**0.9651** (7.20e-04)	14.869

Table 5.14: Results for flat metrics on Fashion MNIST

	Accuracy by Level			Param.
Model	Level 1	Level 2	Level 3	(MM)
Flat CNN [59]	-	-	0.9324 (1.52e-03)	14.739
B-CNN [59]	0.9976 (3.52e-04)	0.9617 (1.79e-03)	0.9316 (1.30e-03)	14.869
BA-CNN	0.9978 (3.63e-04)	**0.9663** (1.72e-03)	0.9313 (2.16e-03)	15.201
H-CNN [58]	0.9978 (3.11e-04)	0.9618 (1.17e-03)	0.9304 (1.50e-03)	15.106
Add-net [19]	0.9978 (2.51e-04)	**0.9631** (2.45e-03)	0.9313 (1.79e-03)	14.869
Concat-net [19]	0.9978 (4.13e-04)	**0.9652** (1.12e-03)	0.9322 (1.35e-03)	14.869

5.7 Time and Space Complexity

In this part, we discuss the proposed model's time and space complexity and the baselines. First, the scalability of the models in terms of the number of parameters depending on the branch's dimension. Then different measures were performed on the models to evaluate the impact of our proposal in comparison with the state-of-the-art models presented in this work.

Regarding scalability, Fig. 5.8 shows the effect of increasing the number of neurons in all branches for the models presented in Section 5.5. The figure shows that all the models increase their space complexity similarly, except H-CNN, which quickly surpasses the rest of the models. It is worth mentioning that this behavior is for the proposed model using 64 neurons in the attention mechanism. If that value increases, the space complexity of the model increases, and, at some point (2048 neurons in this setting), it can reach H-CNN's complexity.

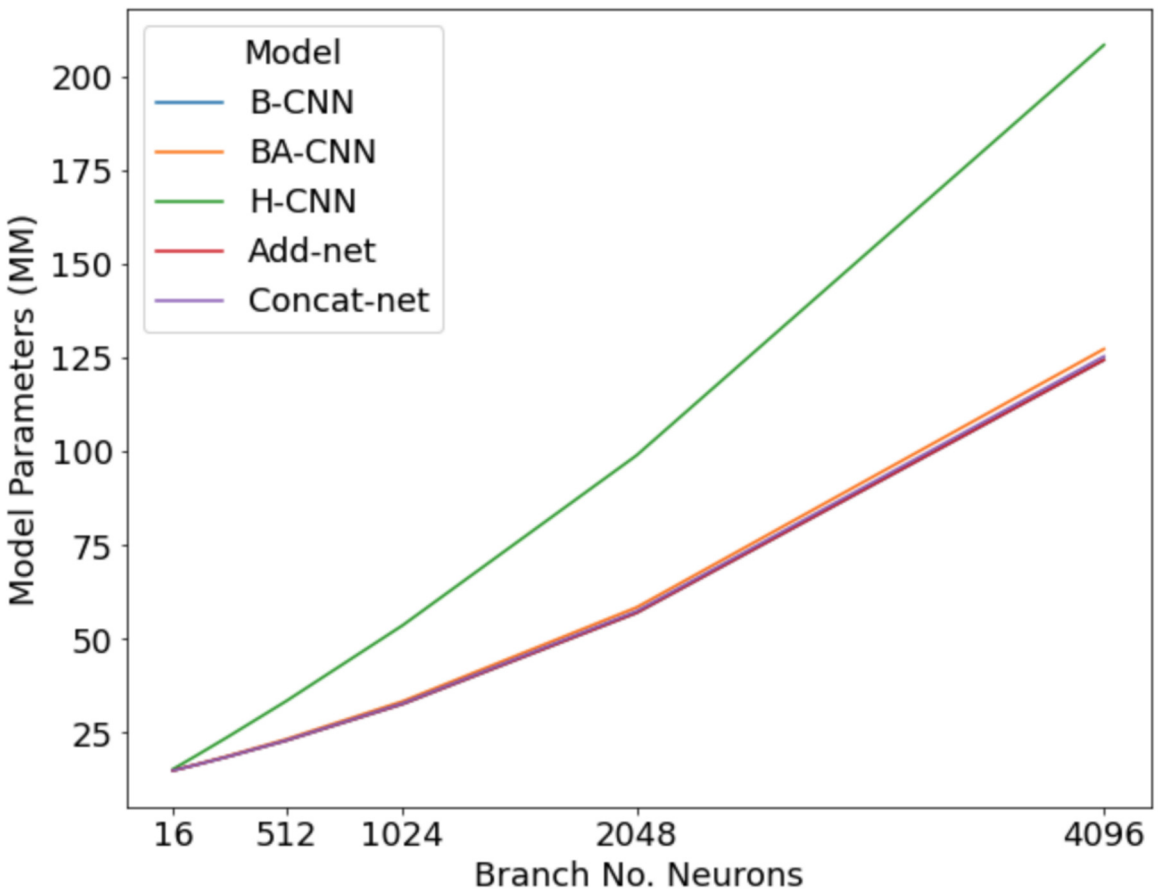


Figure 5.8: Number of model parameters on CIFAR-100 as we increase the number of neurons in the branches.

The following measurements were computed to models presented in Section 5.6 and presented in Table 5.15:

- No. floating point operations per second (FLOPs).
- GPU memory requirement.
- No. model parameters.
- Memory required by model weights.
- Training time.

In terms of training time, considering the fastest model (Flat CNN) as the baseline, the proposed model presented, for all datasets, a less than 1.13x slowdown. Between models, the differences are less than 5%. It is worth noticing that the differences in training time between datasets are primarily due to the number of epochs: 80 for CIFAR-100 and 60 for CIFAR-10 and Fashion MNIST.

The FLOPs measurements confirm that by adopting the same branches' hyper-parameters, the time complexities of the different models are comparable. The main reason for the differences in FLOPs between the datasets is the input size of the network. CIFAR-100 and CIFAR-10 use 32 x 32 images, and Fashion MNIST 28 x 28.

In terms of model parameters and the corresponding memory required, due to the concatenation of branches from the first dense layer, the H-CNN model presents more parameters than the proposed model, excluding Fashion MNIST, always with a tight margin.

In summary, adding the attention mechanisms to a branched architecture does not negatively impact time and space complexity.

Table 5.15: Time and Space Complexity of the models. As a Flat CNN representative we chose VGG-16 which is equivalent to B-CNN without the first two branches

Model	No. FLOPs (MFLOPs)	GPU Memory Requirement (MB)	Model Parameters (MM)	Memory Required by Model Weights (MB)	Training Time (min)
CIFAR-10					
Flat CNN	6.26	301.63	14.76	56.32	14.77 (0.01)
B-CNN	6.27	308.28	15.19	57.96	15.98 (0.09)
BA-CNN	6.29	318.19	15.82	60.36	16.79 (0.01)
H-CNN	6.29	321.91	15.85	60.46	15.80 (0.30)
Add-net	6.27	308.31	15.20	57.96	15.38 (0.03)
Concat-net	6.27	308.45	15.20	57.97	15.37 (0.02)
CIFAR-100					
Flat CNN	6.28	302.94	15.35	58.55	19.93 (0.18)
B-CNN	6.34	313.59	18.64	71.10	21.43 (0.03)
BA-CNN	6.44	329.44	23.43	89.37	22.67 (0.03)
H-CNN	6.45	331.59	23.88	91.10	22.09 (0.03)
Add-net	6.34	313.84	18.64	71.10	21.52 (0.04)
Concat-net	6.34	315.02	18.70	71.33	21.41 (0.25)
Fashion MNIST					
Flat CNN	4.10	224.05	14.74	56.23	14.99 (0.02)
B-CNN	4.11	228.47	14.88	56.75	15.73 (0.01)
BA-CNN	4.11	237.96	15.21	58.02	16.37 (0.26)
H-CNN	4.11	237.32	15.12	57.66	15.22 (0.01)
Add-net	4.11	228.49	14.87	56.75	15.09 (0.02)
Concat-net	4.11	228.55	14.88	56.76	15.13 (0.01)

5.8 Hierarchical Consistency

To visualize the relationships learned by the model, we reconstructed a hierarchy tree from the models presented in Section 5.5[1]. Fig. 5.9, 5.10, 5.11, 5.12 and 5.13 show

[1]Hyperparameter tuning improved the proposed model only; therefore, for a fairer comparison, we decided to use those models.

the relations for CIFAR-10 obtained using the predictions of the proposed model (BA-CNN), B-CNN, H-CNN, Add-net and Concat-net respectively. We can see that the proposed model more precisely reconstructs the correct relations in the hierarchy. Each edge of the graph was drawn using an alpha value proportional to the number of predictions that included that edge.

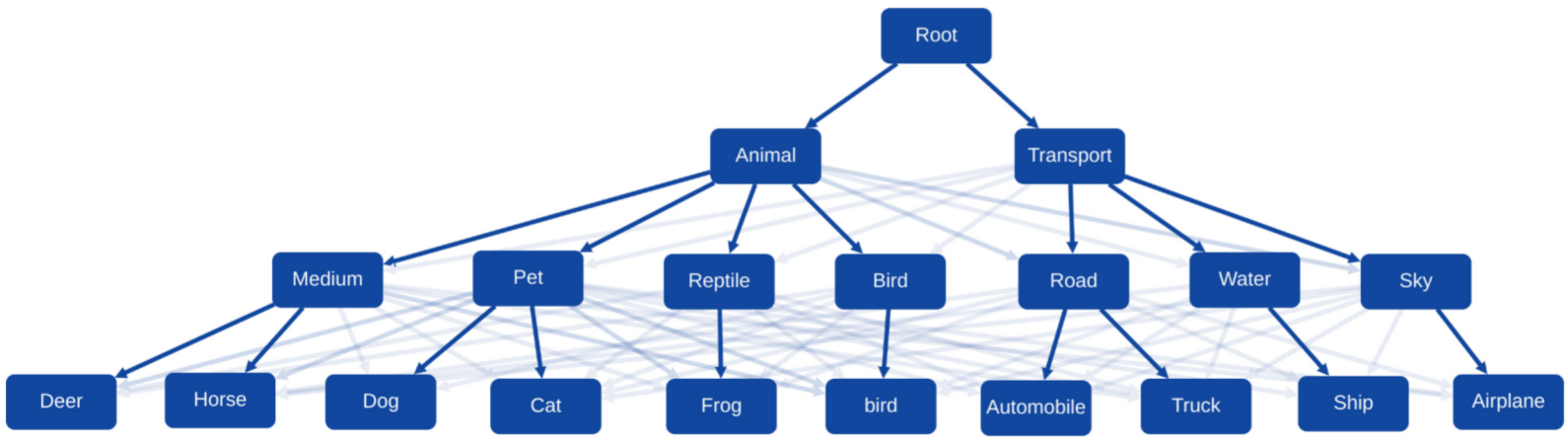


Figure 5.9: Hierarchy tree reconstructed by BA-CNN with BT-Strategy (score = 0.9803).

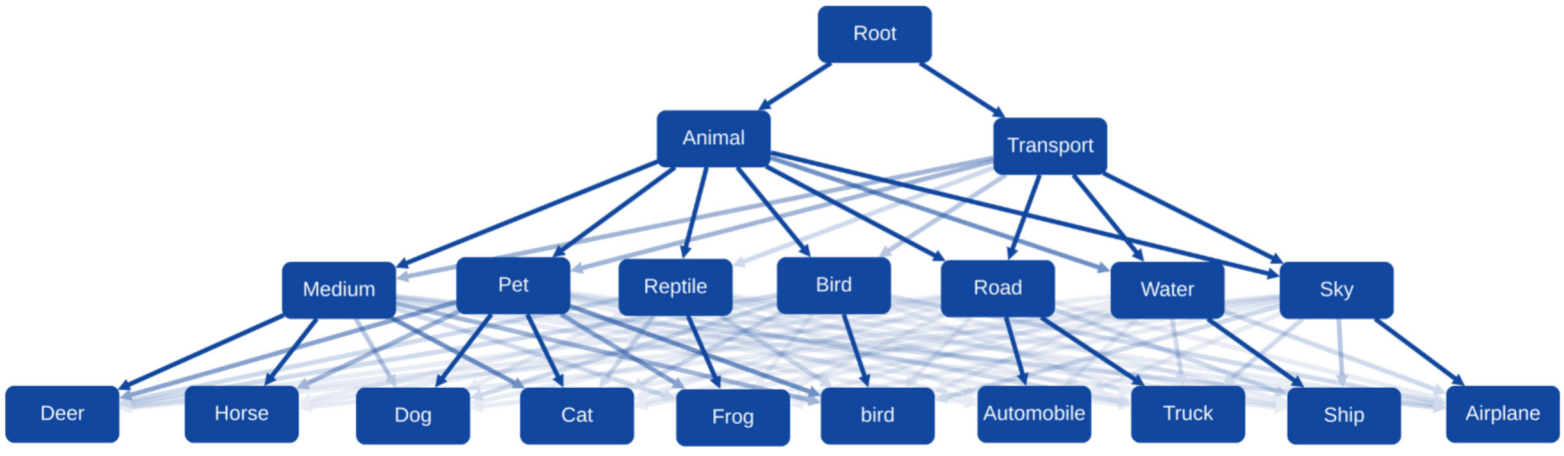


Figure 5.10: Hierarchy tree reconstructed by B-CNN with BT-Strategy (score = 0.8997).

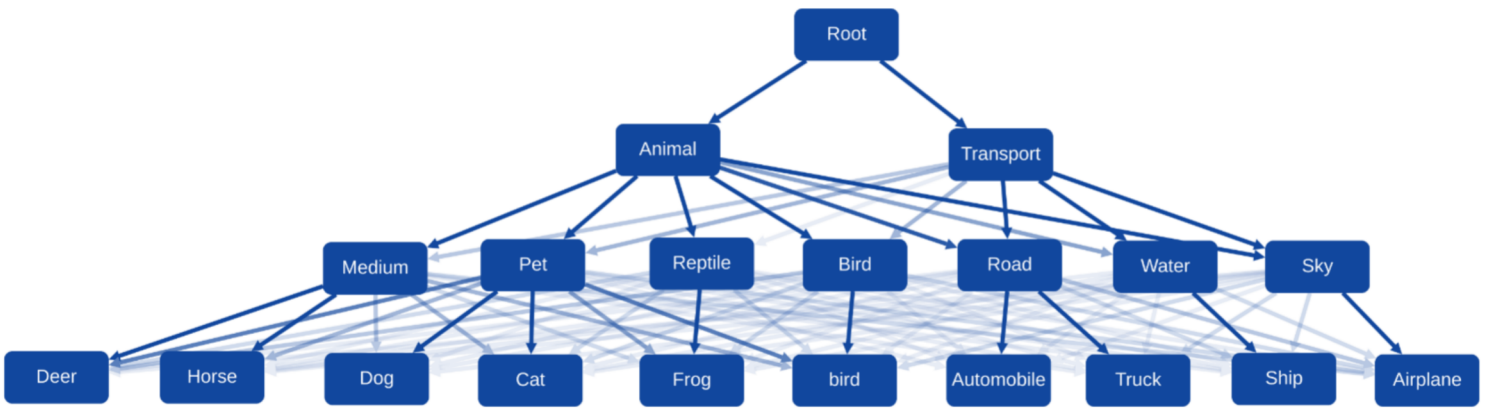


Figure 5.11: Hierarchy tree reconstructed by H-CNN with BT-Strategy (score = 0.9070).

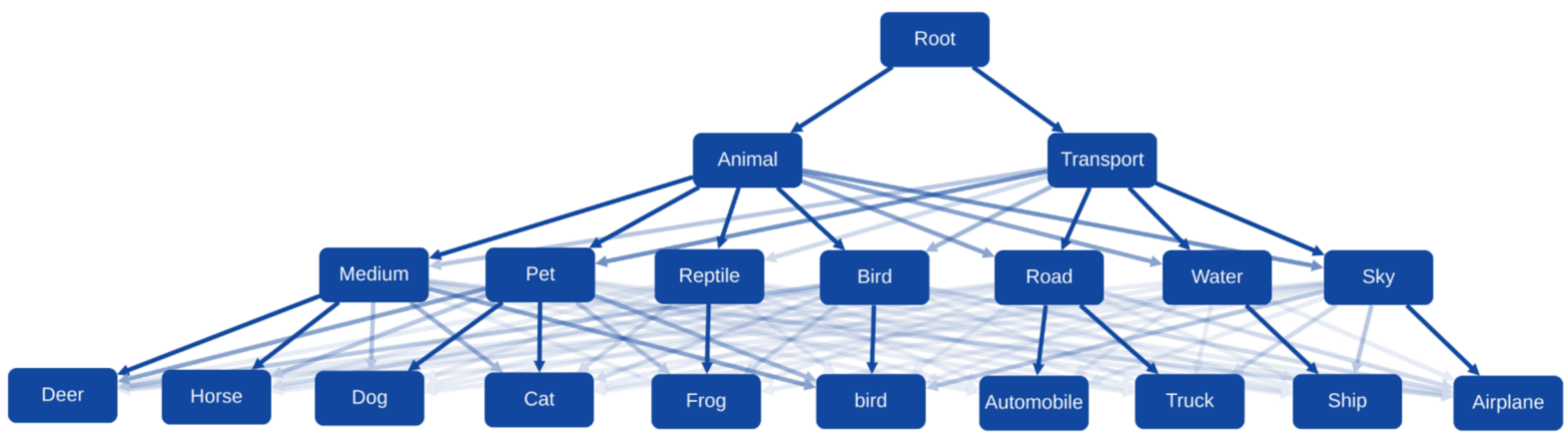


Figure 5.12: Hierarchy tree reconstructed by Add-net with BT-Strategy (score = 0.9212).

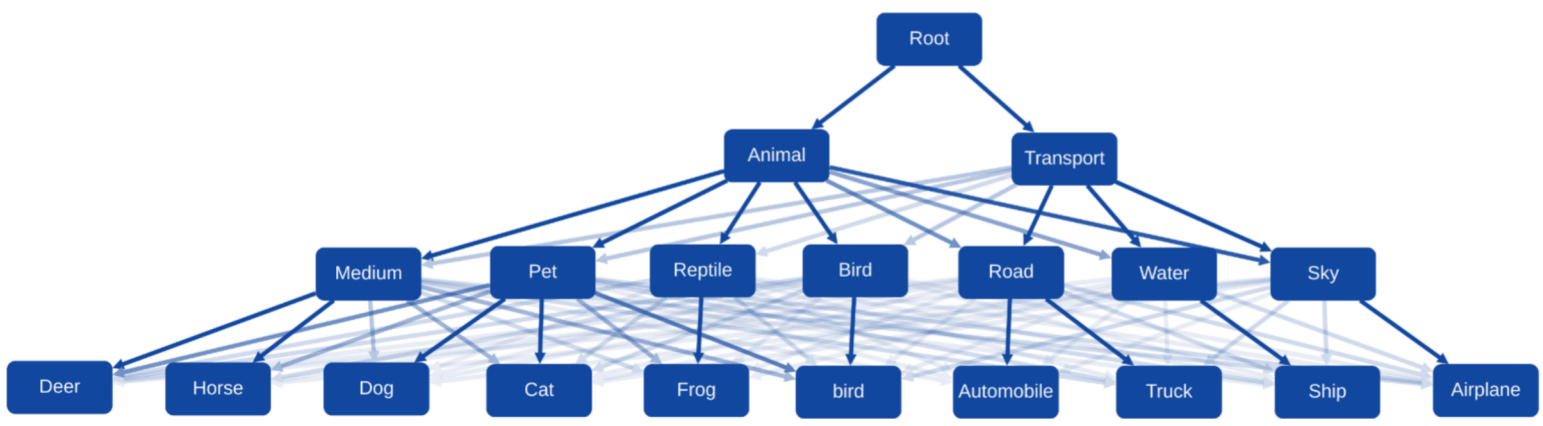


Figure 5.13: Hierarchy tree reconstructed by Concat-net with BT-Strategy (score = 0.9212).

5.9 Attention Weights

In addition to allowing performance gains in many tasks, attention has become a popular method for gaining insights into the internals of deep learning models and their predictions [14]. Indeed, as attention allows the model to focus on a subset of features when predicting an output, attention weights have been used as proxies of feature importance since their early introduction [36]. Bahdanau et al. [5] used attention weights to analyze the alignment between words learned by the model. Xu et al. [56] showed that the attended regions in an image-captioning task aligned well with human intuition. More recently, Voita et al. [50] demonstrated that the maximum attention weight of a head in Transformer models agrees reasonably well with its relevance to machine-translation tasks. Although some recent studies have suggested that attention scores can be inconsistent with other feature importance measures [20, 43], subsequent experiments have shown that attention weights can still help analyze the decision and learning mechanisms of otherwise black-box models [53, 47, 14].

Here, we briefly analyze how the proposed model attends to different levels of the class hierarchy during training. For a given level s, we compute the attention weight α_{st} given to branch s when classifying data at level t and report the average among the training instances. Fig. 5.14 depicts the resulting scores for CIFAR-10. We can see that the coarser levels ($t = 1$ and $t = 2$) give much more importance to features extracted by branch $s = 3$. Attention weights are not constant during training but quickly converge to a configuration in which features extracted from coarser branches have negligible importance. This result suggests that *bottom-up feedback* between branches is useful for learning a class taxonomy, a type of internal feedback that solutions have systematically ignored, thus favoring top-down communication patterns. Fig. 5.16 confirms this result for CIFAR-100. Although the convergence is slightly slower, Fig. 5.17 shows that the same pattern is also observed for level $t = 2$ on Fashion MNIST. Interestingly, however, level $t = 1$ spreads attention more evenly among the branches, suggesting that the attention distribution is case-dependent: fixed attention weights can work in some datasets but produce unsatisfactory results in others. Fig. 5.14, 5.16, and 5.17 show that attention at level $t = 3$ often behaves differently from the other levels and varies more among the datasets. In addition, these figures show that the effect of the BT-Strategy is to systematically increase the attention the branches give to the finer level of the class hierarchy, which is expected because the BT-Strategy gradually focuses training on that branch.

In Fig. 5.15, 5.18, and 5.19, we show how the attention distribution varies among different classes. For this experiment, we averaged the attention weights by considering only (training) instances belonging to the same class at a given level t of the hierarchy. We selected cases that depicted the variety of attention patterns we observed in our experiments. For CIFAR-10, for instance, as shown in Fig. 5.15, there are three classes at level $t = 3$, which contradicts the global attention distribution

previously discussed. For these classes, the attention mechanism weights more the feature vector received from the same branch ($s = 3$). We can also observe in Fig. 5.15 that the attention given to the previous branch ($s = 2$) varies significantly between classes. In Fig. 5.18 and 5.19, we present the attention weights for two classes of the first level and two classes at the last level. We can see that the relevance of branch $s = 2$ for the predictions at level $t = 1$ can respect the global pattern (class "Clothes") or be significantly higher than the average curves suggest (class "Goods"). Similarly, at level $t = 3$, we can find classes for which the attention given to the feature vector from the same branch ($s = 3$) is the lowest, contradicting the general behavior. We offer these results as evidence of the fact that the context information that the model needs to disambiguate a class varies from case to case. This supports the hypothesis that an attention mechanism among branches can be more effective for hierarchical classification than a fixed connectivity pattern or a handcrafted rule to feed the branches.

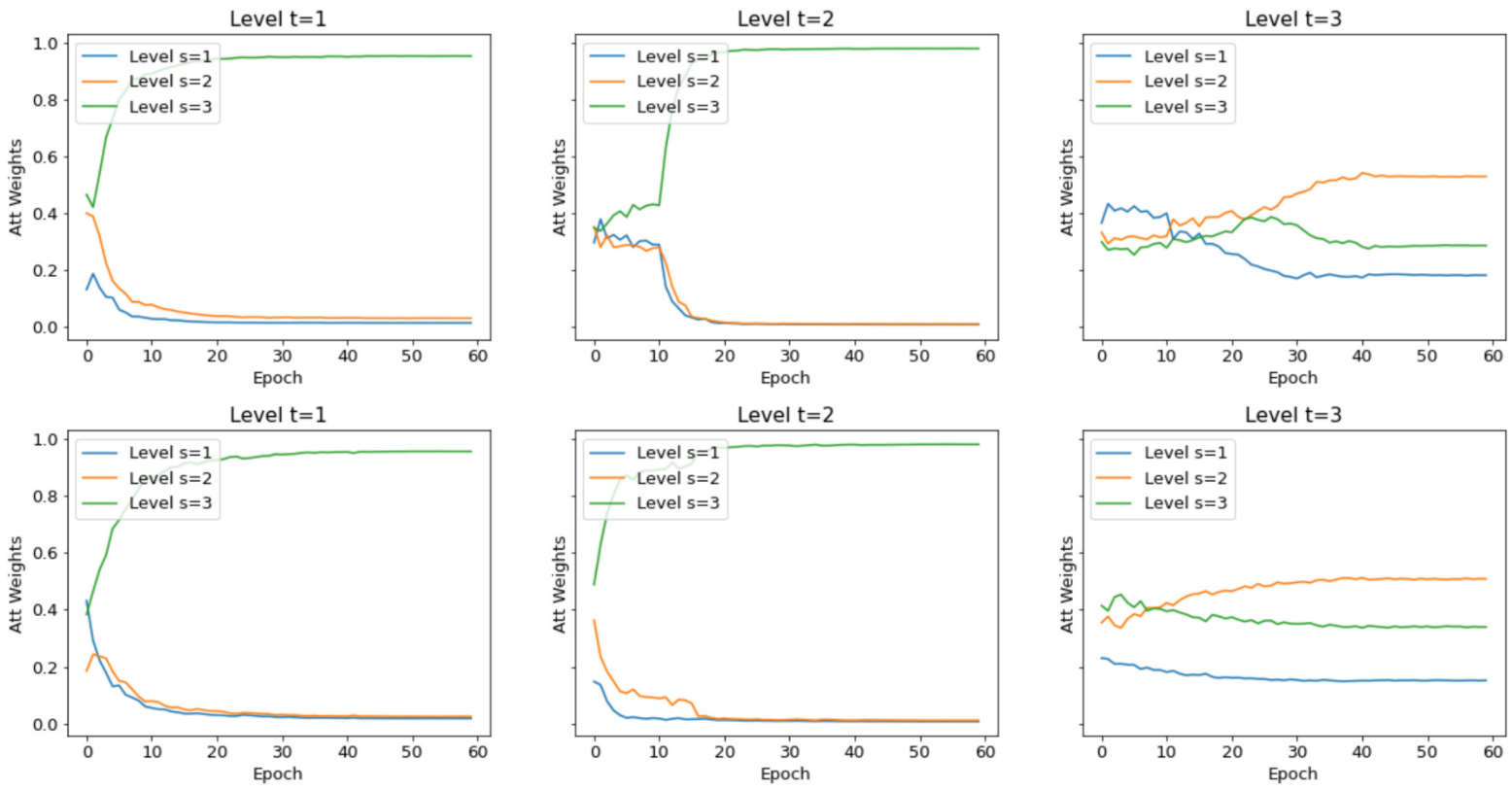


Figure 5.14: Attention weights per level on CIFAR-10. On the first row, using BT-Strategy, and in the second row without it.

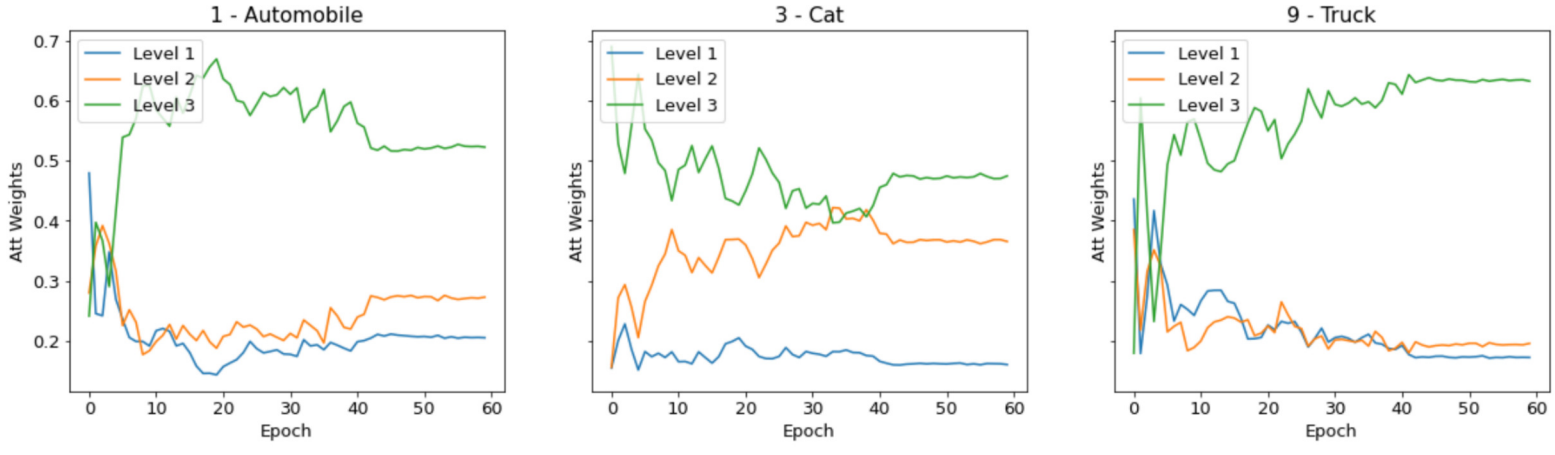


Figure 5.15: Attention weights from three classes on CIFAR-10 for t=3.

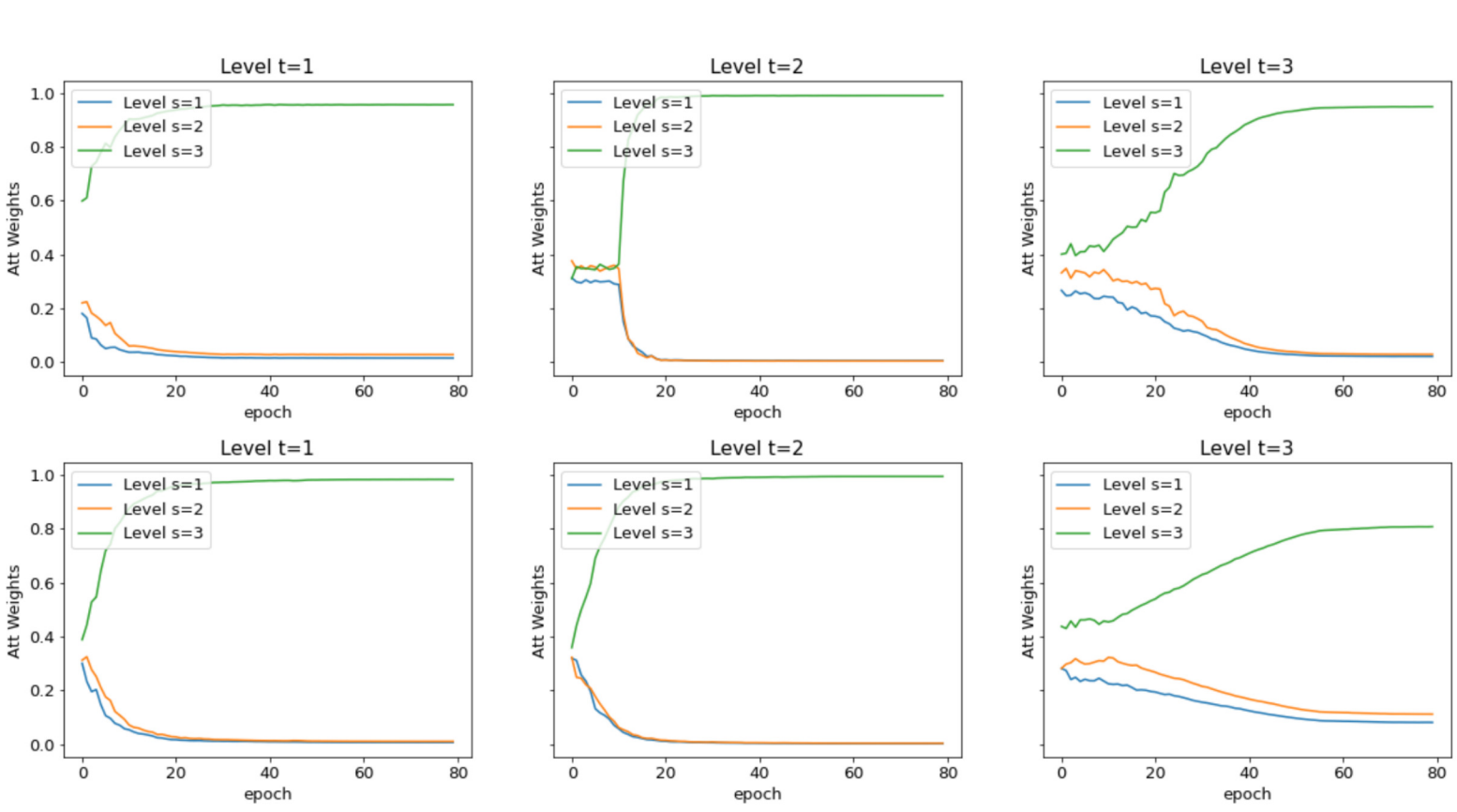


Figure 5.16: Attention weights per level on CIFAR-100. On the first row, using BT-Strategy, and in the second row without it.

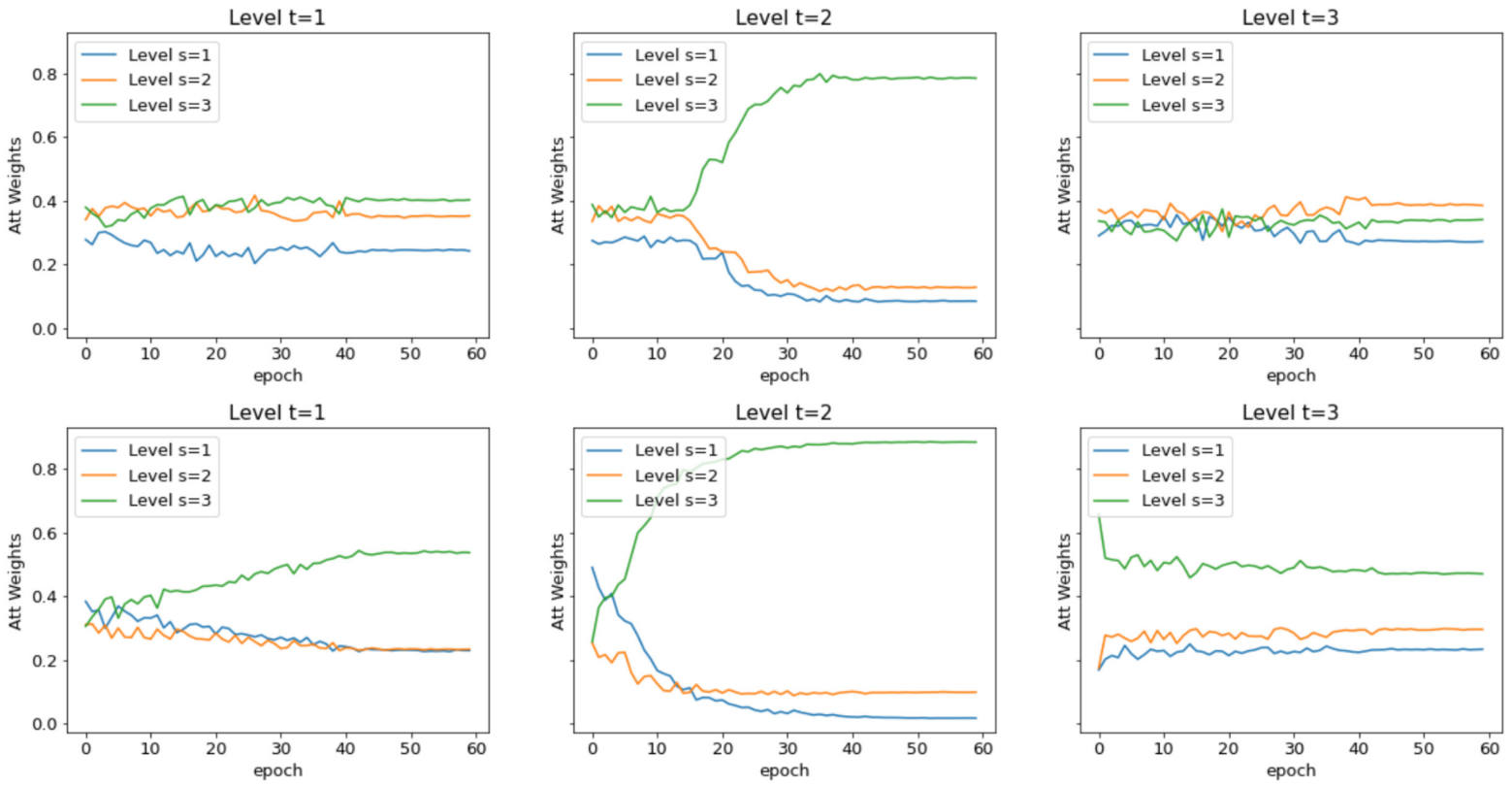


Figure 5.17: Attention weights per level on Fashion MNIST. On the first row, using BT-Strategy, and in the second row without it.

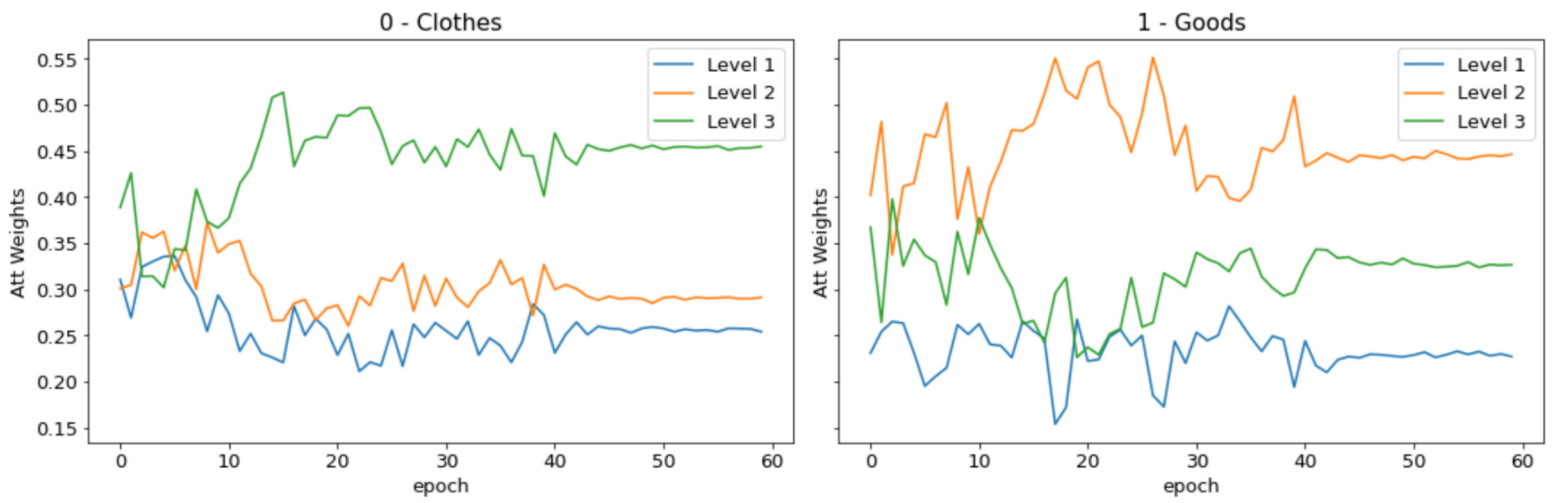


Figure 5.18: Attention weights for level 1 on Fashion MNIST.

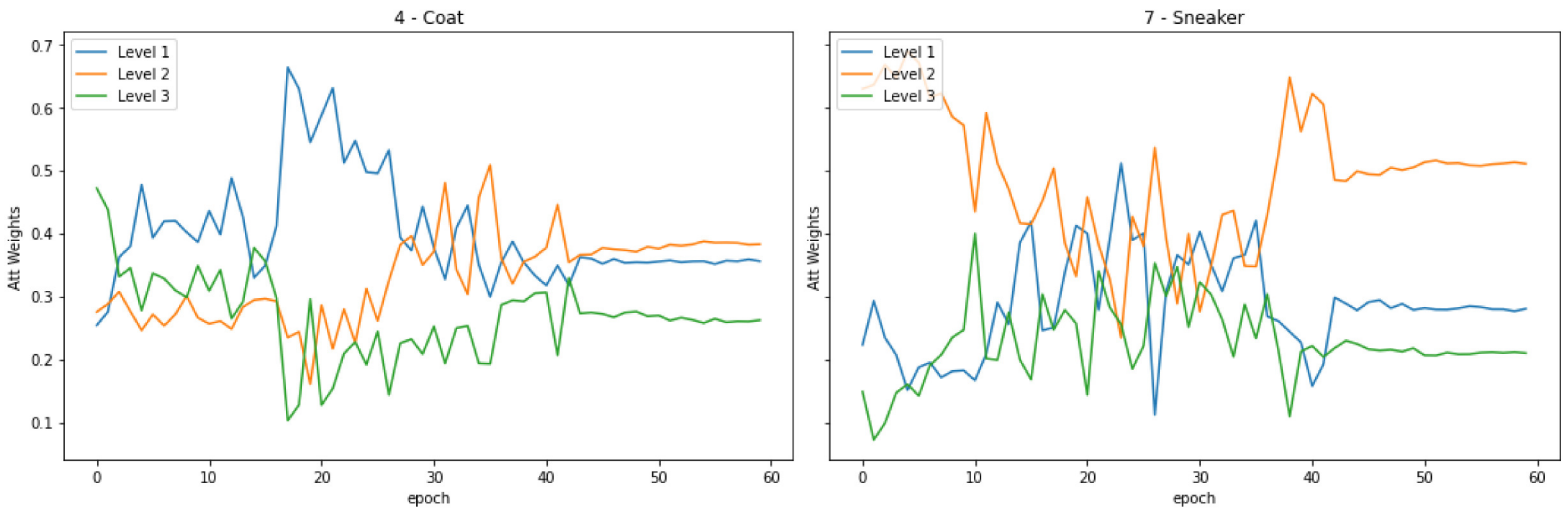


Figure 5.19: Attention weights for class "Coat" and "Sneaker" from Fashion MNIST dataset.

5.10 Trade-off Analysis

As we observed in previous sections, the addition of the attention mechanism to the branched CNN outperformed state-of-the-art models in terms of hierarchical metrics. The above translates into more consistent and accurate predictions for hierarchical classification problems. We attribute these results to the dynamic connection between branches performed by the attention mechanism, which has been confirmed through the analysis of the attention weights. We note that this dynamism is required across the different hierarchy levels, classes, and datasets.

However, these improvements come with a trade-off. In some cases, the proposed model maintains or slightly underperforms the state-of-the-art models in the last level of the hierarchy. This limitation can be mitigated by preprocessing the input data or using a specific schedule for BT-Strategy.

Finally, it is important to highlight that although the conducted experiments did not reveal any significant disadvantages in terms of time and memory footprint, the computational cost will inevitably increase as the number B of hierarchy levels increases. This is due to the proposed attention mechanism having quadratic complexity in B. Therefore, when dealing with deep class hierarchies, a careful consideration of the available computational resources is necessary.

Chapter 6

Conclusions and future work

This study presented a novel technique for deep hierarchical classification that recombines feature maps extracted at different depths of a convolutional neural network to classify data into the different levels of the class hierarchy. The novelty of our approach lies in introducing an attention mechanism devised to learn how the predictions at different hierarchy levels must influence each other during training and inference. The attention module allows feature maps to be exchanged and fused in a dynamic data-driven way that supports, in particular, top-down and bottom-up cross-level interactions.

We assessed the proposed methodology using well-known hierarchical benchmarks based on the CIFAR-10, CIFAR-100, and Fashion MNIST datasets. Experiments demonstrated that, in all cases, our algorithm improves the hierarchical accuracy and hierarchical consistency of four state-of-the-art models in a statistically significant way. Regarding per-level performance, we found that the coarser taxonomic levels are typically those that benefit most from the attention mechanism. At the deeper level of the hierarchy, we observed more mixed results, but statistical tests reveal that competitive predictions for that specific level also come out very often. For instance, in the CIFAR-100 dataset, the proposed method consistently outperformed existing methods by a margin of around 10% accuracy at the first level of the hierarchy, at the cost of only about 4% accuracy at the deeper hierarchy level. This result is remarkable because CIFAR-100 has a large number of classes.

To conclude, we designed experiments to assess the hypothesis that our method is better than baselines for learning hierarchical dependencies between concepts. Experiments showed that our technique outperforms state-of-the-art models in hierarchical consistency: it typically traverses the hierarchy by choosing consistent paths from the root to the leaves. In particular, we observed the most considerable improvement in the CIFAR-100 dataset, where the proposed method achieved a 25% improvement on current methods.

In view of the experimental results, we thus conclude that our method advances current research on deep hierarchical classification providing predictions more consistent with the class hierarchy. Furthermore, the analysis of the attentional weights

revealed that hierarchical classification benefits from the dynamic connection between branches provided by the attention mechanism. The neural net updates the importance of the input feature maps as training progresses and uses different weights to classify different input patterns. This flexibility contrasts with current solutions' rigid and sometimes cumbersome connectivity patterns. In addition, this flexibility allows the model to use a low number of parameters to obtain the current advantages, which translates only in a slight impact in terms of time and space complexity.

In current methods (including ours), the depth at which a branch emerges is handcrafted by the modeler. Future work will explore the idea that the network can automatically and dynamically make that choice. Specifically, we plan to devise a secondary attention mechanism that can select and aggregate visual features from different depths of the central block to feed the branches. We plan to investigate also different implementations of the primary attention mechanism. In particular, incorporating recent advances in deep multi-head attention is an exciting direction to explore.

Bibliography

[1] P. Aggarwal. Fashion product images dataset. *Retrieved from kaggle: https://www. kaggle. com/paramaggarwal/fashion-product-images-dataset*, 2019.

[2] G. An, M. Akiba, K. Omodaka, T. Nakazawa, and H. Yokota. Hierarchical deep learning models using transfer learning for disease detection and classification based on small number of medical images. *Scientific reports*, 11(1):1–9, 2021.

[3] K. Atkinson, T. Bench-Capon, and D. Bollegala. Explanation in ai and law: Past, present and future. *Artificial Intelligence*, 289:103387, 2020.

[4] R. Babbar, I. Partalas, E. Gaussier, and M. R. Amini. On flat versus hierarchical classification in large-scale taxonomies. *Advances in neural information processing systems*, 26, 2013.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[6] J. G. s. Barbedo and A. Lopes. Automatic genre classification of musical signals. *EURASIP Journal on Advances in Signal Processing*, 2007:1–12, 2006.

[7] Z. Barutcuoglu, R. E. Schapire, and O. G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[8] A. Bilal, A. Jourabloo, M. Ye, X. Liu, and L. Ren. Do convolutional neural networks learn class hierarchy? *IEEE transactions on visualization and computer graphics*, 24(1):152–162, 2017.

[9] J. J. Burred and A. Lerch. A hierarchical approach to automatic musical genre classification. In *Proceedings of the 6th international conference on digital audio effects*, pages 8–11, 2003.

[10] R. Cerri, R. C. Barros, and A. C. de Carvalho. Hierarchical classification of gene ontology-based protein functions with neural networks. In *2015 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2015.

[11] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 577–585, Cambridge, MA, USA, 2015. MIT Press.

[12] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263, 2000.

[13] A. A. Freitas and A. C. de Carvalho. *Research and Trends in Data Mining Technologies and Applications*, chapter A Tutorial on Hierarchical Classification with Applications in Bioinformatics. Idea Group Pub., 2007.

[14] A. Galassi, M. Lippi, and P. Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4291–4308, 2020.

[15] D. Gao. Deep hierarchical classification for category prediction in E-commerce system. In *Proceedings of The 3rd Workshop on e-Commerce and NLP*, pages 64–68, Seattle, WA, USA, July 2020. Association for Computational Linguistics.

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

[17] M. Guo, T. Xu, J. Liu, Z. Liu, P. Jiang, T. Mu, S. Zhang, R. R. Martin, M. Cheng, and S. Hu. Attention mechanisms in computer vision: A survey. *Computational Visual Media*, 2022.

[18] J. Hernández, L. E. Sucar, and E. F. Morales. Multidimensional hierarchical classification. *Expert systems with applications*, 41(17):7671–7677, 2014.

[19] M. Inoue, C. H. Forster, and A. C. dos Santos. Semantic hierarchy-based convolutional neural networks for image classification. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[20] S. Jain and B. C. Wallace. Attention is not explanation. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 3543–3556. Association for Computational Linguistics, 2019.

[21] S. Kiritchenko, S. Matwin, and A. F. Famili. Functional annotation of genes using hierarchical text categorization. In *in Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05)*, 2005.

[22] S. Kiritchenko, S. Matwin, R. Nock, and A. F. Famili. Learning and evaluation in the presence of class hierarchies: Application to text categorization. In L. Lamontagne and M. Marchand, editors, *Advances in Artificial Intelligence*, pages 395–406, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[23] B. Kolisnik, I. Hogan, and F. Zulkernine. Condition-cnn: A hierarchical multi-label fashion image classification model. *Expert Systems with Applications*, 182:115195, 2021.

[24] A. Kosmopoulos, I. Partalas, E. Gaussier, G. Paliouras, and I. Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, 29:820–865, 2015.

[25] K. Kowsari, R. Sali, M. Khan, W. Adorno, S. Ali, S. Moore, B. Amadi, P. Kelly, S. Syed, and D. Brown. Diagnosis of celiac disease and environmental enteropathy on biopsy images using color balancing on convolutional neural networks. In *Proceedings of the Future Technologies Conference (FTC 2019)*, 2019.

[26] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[28] Z. Kuang, J. Yu, Z. Li, B. Zhang, and J. Fan. Integrating multi-level deep learning and concept ontology for large-scale visual recognition. *Pattern Recognition*, 78:198–214, 2018.

[29] R. La Grassa, I. Gallo, and N. Landro. Learn class hierarchy using convolutional neural networks. *Applied Intelligence*, 51:6622–6632, 2021.

[30] Y. Labrou and T. Finin. Yahoo! as an ontology: Using yahoo! categories to describe documents. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, page 180–187, New York, NY, USA, 1999. Association for Computing Machinery.

[31] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[32] C.-C. Lee, E. Mower, C. Busso, S. Lee, and S. Narayanan. Emotion recognition using a hierarchical binary decision tree approach. *Speech Communication*, 53(9-10):1162–1171, 2011.

[33] J. Lee, J.-H. Shin, and J.-S. Kim. Interactive visualization and manipulation of attention-based neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 121–126, 2017.

[34] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1457, 2015.

[35] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics.

[36] Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[37] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[38] J. J. Rocchio. *Relevance Feedback in Information Retrieval*, pages 313–323. Prentice Hall, Englewood, Cliffs, New Jersey, 1971.

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[40] R. Sali, S. Adewole, L. Ehsan, L. A. Denson, P. Kelly, B. C. Amadi, L. Holtz, S. A. Ali, S. R. Moore, S. Syed, and D. E. Brown. Hierarchical deep convolutional neural networks for multi-category diagnosis of gastrointestinal disorders on histopathological images. In *2020 IEEE International Conference on Healthcare Informatics (ICHI)*, pages 1–6, 2020.

[41] C. Sammut and G. I. Webb, editors. *Accuracy*, pages 9–10. Springer US, Boston, MA, 2010.

[42] Y. Seo and K.-s. Shin. Hierarchical convolutional neural networks for fashion image classification. *Expert Systems with Applications*, 116:328–339, 2019.

[43] S. Serrano and N. A. Smith. Is attention interpretable? In A. Korhonen, D. R. Traum, and L. Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2931–2951. Association for Computational Linguistics, 2019.

[44] C. N. Silla and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72, 2011.

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[46] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 521–528, 2001.

[47] M. Tutek and J. Šnajder. Toward practical usage of the attention mechanism as a tool for interpretability. *IEEE Access*, 10:47011–47030, 2022.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[49] C. Vens, J. Struyf, L. Schietgat, et al. Decision trees for hierarchical multi-label classification. *Mach Learn*, 73(185), 2008.

[50] E. Voita, D. Talbot, F. Moiseev, I. Titov, and R. Sennrich. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 5797–5808, 2020.

[51] X.-L. Wang, H. Zhao, and B.-L. Lu. A meta-top-down method for large-scale hierarchical classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):500–513, 2013.

[52] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*, pages 499–515. Springer, 2016.

[53] S. Wiegreffe and Y. Pinter. Attention is not not explanation. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 11–20. Association for Computational Linguistics, 2019.

[54] F. Wu, J. Zhang, and V. Honavar. Learning classifiers using hierarchically structured class taxonomies. In J.-D. Zucker and L. Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 313–320, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[55] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[56] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[57] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 2740–2748, 2015.

[58] X. Zhang, L. Tang, H. Luo, S. Zhong, Z. Guan, L. Chen, C. Zhao, J. Peng, and J. Fan. Hierarchical bilinear convolutional neural network for image classification. *IET Computer Vision*, 15(3):197–207, 2021.

[59] X. Zhu and M. Bain. B-cnn: branch convolutional neural network for hierarchical classification. *arXiv preprint arXiv:1709.09890*, 2017.

Appendix A

Tools and Technical Specifications

Here are listed the several tools and environments for software developing and testing that were used during this work.

A.1 Environment

All the models were implemented using Python with the functional Deep Learning API Keras.

A.2 Hardware

Specific hardware (CPU, RAM, HDD and VRAM) for each environment is listed at Table A.1.

Table A.1: Hardware specifications for work environment

Personal Computer	
CPU	AMD Ryzen 7 5700G @3.8GHZ
RAM	32 GB
HDD	500 GB
GPU	NVIDIA GeForce RTX 3060ti
VRAM	8GB

Appendix B

Statistical Tests Results

This Appendix includes all the p-values from the Friedman and Wilcoxon tests performed on the related experiments. It is important to note that the Wilcoxon test was performed against the proposed model. When our model did not obtain the best numerical result, we performed that test against the winner and presented the results in complementary tables. The above occurs in Tables B.2, B.5, B.8, B.11, and B.14. P-values are highlighted when the statistical test shows significant differences at 5% level (p-value > 0.05).

B.1 P-values for Experiment 1

B.1.1 CIFAR-10

Table B.1: P-values of the Friedman test, by groups on CIFAR-10

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base B, w/BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Base C, w/BT	0.0000	0.0000	0.0004	0.0000	0.0000	0.0000
Base B, w/o BT	0.0000	0.0006	0.0465	0.0000	0.0000	0.0001
Base C, w/o BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table B.2: P-values of the Wilcoxon Test, against BA-CNN on CIFAR-10

	Accuracy by Level			**Hierarchical Metrics**		
Model Base-B	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Model Base-C						
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Without BT-Strategy Model Base-B						
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	**1.0000**	-	0.0195	0.0195	**0.1367**
Add-net	0.0195	**0.0586**	-	0.0195	0.0195	0.0391
Concat-net	0.0195	**0.0977**	-	0.0195	0.0195	**0.0586**
Model Base-C						
B-CNN	0.0195	0.0195	**0.0586**	0.0195	0.0195	0.0195
H-CNN C	0.0195	0.0195	**0.0879**	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	**1.0000**	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	**0.4102**	0.0195	0.0195	0.0195

Table B.3: P-values of the Wilcoxon Test on CIFAR-10 (Complementary table of Table B.2)

Model Base-B, against B-CNN	Accuracy on Level 3
BA-CNN	**1.0000**
H-CNN	**1.0000**
Add-net	0.0293
Concat-net	0.0293
Model Base-C, against B-CNN	
BA-CNN	**1.0000**
H-CNN	0.0293
Add-net	0.0293
Concat-net	0.0486
Without BT-Strategy Model Base-B, against H-CNN	
B-CNN	**0.5566**
BA-CNN	**0.1465**
Add-net	**1.0000**
Concat-net	**1.0000**

B.1.2 CIFAR-100

Table B.4: P-values of the Friedman test, by groups on CIFAR-100.

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base C, w/BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Base C, w/o BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table B.5: P-values of the Wilcoxon test against BA-CNN on CIFAR-100

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Without BT-Strategy						
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195

Table B.6: P-values of the Wilcoxon Test on CIFAR-100 (Complementary table of Table B.5)

Against H-CNN	Accuracy on Level 3
B-CNN	0.0293
BA-CNN	0.0293
Add-net	0.0293
Concat-net	0.0293

Without BT-Strategy Against H-CNN	
B-CNN	0.0293
BA-CNN	0.0293
Add-net	0.0293
Concat-net	0.0293

B.1.3 Fashion MNIST

Table B.7: P-values of the Friedman test, by groups on Fashion MNIST.

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base C, w/BT	**0.0915**	0.0025	0.0009	0.0001	0.0000	0.0072
Base C, w/o BT	**0.6225**	0.0020	0.0014	0.0000	0.0000	0.0027

Table B.8: P-values of the Wilcoxon test against BA-CNN on Fashion MNIST

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	**1.0000**	**0.1367**	-	0.0195	0.0195	-
H-CNN	**0.1591**	**1.0000**	-	0.0391	0.0195	-
Add-net	**0.0591**	**0.2734**	-	0.0195	0.0195	-
Concat-net	**1.0000**	0.0195	-	0.0195	0.0195	-
Without BT-Strategy						
B-CNN	**1.0000**	-	**0.2930**	0.0195	0.0195	-
H-CNN	**1.0000**	-	**1.0000**	0.0195	0.0195	-
Add-net	**1.0000**	-	**0.5566**	0.0195	0.0195	-
Concat-net	**1.0000**	-	**0.9668**	0.0195	0.0195	-

Table B.9: P-values of the Wilcoxon Test on Fashion MNIST (Complementary table of Table B.8)

Against H-CNN	Accuracy on Level 3	Hierarchical F1
B-CNN	**0.0586**	0.0195
BA-CNN	**0.2930**	**0.4883**
Add-net	**0.2051**	**0.0586**
Concat-net	**0.2051**	0.0391

Without BT-Strategy

Against H-CNN	Accuracy on Level 2	Hierarchical F1
B-CNN	**0.0977**	**0.1367**
BA-CNN	**1.0000**	**1.0000**
Add-net	0.0195	**0.0977**
Concat-net	0.0391	**0.1953**

B.2 P-values for Experiment 2

B.2.1 CIFAR-10

Table B.10: P-values of the Friedman test, by groups on CIFAR-10 for experiment 2.

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base C, w/BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table B.11: P-values of the Wilcoxon Test, against BA-CNN on CIFAR-10 for experiment 2

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195

Table B.12: P-values of the Wilcoxon Test on CIFAR-10 for experiment 2 (Complementary table of Table B.11)

Against B-CNN	Accuracy on Level 3
BA-CNN	**0.0593**
H-CNN	**1.0000**
Add-net	0.0293
Concat-net	0.0293

B.2.2 CIFAR-100

Table B.13: P-values of the Friedman test, by groups on CIFAR-100 for experiment 2.

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base C, w/BT	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table B.14: P-values of the Wilcoxon test against BA-CNN on CIFAR-100 for experiment 2

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
H-CNN	0.0195	0.0195	-	0.0195	0.0195	0.0195
Add-net	0.0195	0.0195	-	0.0195	0.0195	0.0195
Concat-net	0.0195	0.0195	-	0.0195	0.0195	0.0195

Table B.15: P-values of the Wilcoxon Test on CIFAR-100 for experiment 2 (Complementary table of Table B.14)

Against flatt	Accuracy on Level 3
B-CNN	**0.1564**
BA-CNN	0.0293
H-CNN	0.0293
Add-net	0.0293
Concat-net	**0.1464**

B.2.3 Fashion MNIST

Table B.16: P-values of the Friedman test, by groups on Fashion MNIST for experiment 2

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
Base C, w/BT	**0.6262**	0.0000	**0.0888**	0.0000	0.0000	0.0002

Table B.17: P-values of the Wilcoxon test against BA-CNN on Fashion MNIST for experiment 2

	Accuracy by Level			**Hierarchical Metrics**		
Model	Level 1	Level 2	Level 3	Accuracy	Consistency	F1
B-CNN	**1.0000**	**0.0592**	**1.0000**	0.0195	0.0195	**0.1953**
H-CNN	**1.0000**	0.0195	**1.0000**	0.0195	0.0195	0.0195
Add-net	**1.0000**	**0.1083**	**1.0000**	0.0195	0.0195	**1.0000**
Concat-net	**1.0000**	**1.0000**	**1.0000**	**0.0977**	0.0195	**1.0000**