Learning Semantics of Classes in Image Classification

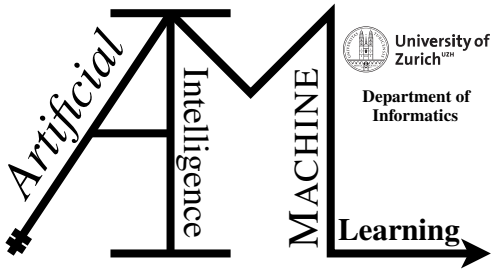
Attention-Sharing between Hierarchies

Master Thesis

Raffael Mogicato
18-742-767

Submitted on
August 8, 2023

Thesis Supervisor
Prof. Dr. Manuel Günther



Master Thesis

Author: Raffael Mogicato, raffael.mogicato@uzh.ch

Project period: 08.02.2023 - 08.08.2023

Artificial Intelligence and Machine Learning Group
Department of Informatics, University of Zurich

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr. Manuel Günther for his guidance during this thesis. Without his input and expertise, this thesis would not have been possible. On a more personal note, I would like to thank my girlfriend Zoe and my family for their continued support. Special thanks to my friends Alain and Adrian for their constructive critiques and insights on my work.

Abstract

Deep convolutional neural networks (CNNs) have become the state-of-the-art approach for image classification. While these networks are very effective at identifying the class to which an image belongs, they often do not properly learn the semantic relationship between classes. This means that models treat all misclassifications equally during training, regardless of the semantic distance between the predicted and actual class. This approach does not reflect the complexity of the real world, where some entities are more similar to each other, making mistakes between related classes less severe than those of unrelated classes. An architecture suited for hierarchical classification is presented as a potential solution to this problem. Rather than just predicting a single class, networks predict a simplified hierarchy consisting of higher-level concepts.

This thesis explores how the architecture of CNNs can be adapted to incorporate hierarchical information to increase performance and the semantical conditioning of CNNs. The ultimate goal is to enhance the accuracy and robustness of image classification models by improving their understanding of the semantic relationships between classes, which could potentially lead to fewer and less severe misclassifications.

To achieve this, several architectures are explored – all using a ResNet backbone with classifiers for each hierarchical level – that are compared with a baseline model that does not utilize the hierarchy for predictions. Most importantly, this thesis proposes an attention mechanism that does not contain any extra trainable parameters. This attention mechanism transforms the deep features given to a lower-level classifier based on the weight matrix from the higher-level classifier. This transformation aims to highlight features relevant to the classification of the higher-level concept, thus enabling the model to learn the decision boundary between classes of different higher-level concepts.

This attention mechanism can effectively increase the classification accuracy for the ImageNet classes compared to a baseline architecture. Furthermore, when provided ground-truth information about the hierarchies from classes during training, it effectively learns the decision boundaries between classes from different higher-level concepts. This thesis also explores whether these architectures can be used for open-set classification. While showing some potential, the attention mechanism could likely be adapted for open-set classification, representing a promising possibility for future research.

Contents

Chapter 1

Introduction

Deep learning has emerged as a powerful tool in the toolbox of artificial intelligence research, presenting unprecedented capabilities, especially in the realm of image classification. The essence of this approach is typically a neural network that uses multiple convolutional layers followed by one or more fully-connected (FC) layers. The convolutional layers create a spatially-aware deep feature map, which is then classified by the FC layer into different classes. Convolutional neural networks (CNNs) learn these classes by predicting a probability distribution for a given image and comparing it to the ground truth label. With this methodology, great performance on many benchmark data sets has been achieved. While such models achieve good results, they do not necessarily grasp the semantic relationship between classes. Research has shown that deep features extracted from pre-trained models can be effectively applied to related tasks (Kornblith et al., 2019), suggesting the presence of some degree of semantic understanding. This highlights the potential for improved exploration and utilization of this aspect of model learning.

Still, the general lack of semantic consideration means that models treat all misclassification equally during training, regardless of the semantic distance between the predicted and actual class. For instance, if a model misclassifies an image of a *German Shepherd* as an *Italian Greyhound*, it is penalized the same way as if it misclassifies a *German Shepherd* as an *airplane*. This happens despite the former mistake being more understandable due to the semantic similarity between different breeds of dogs. This results from the fact that the data set, and thus the model, assumes only a single correct answer, and all other answers are equally wrong. While this approach is popular for a good reason, namely that it allows for a simplified model of reality, it does not properly reflect the complexity of the real world. In reality, some entities are more similar to each other, making mistakes between related classes less severe than those of unrelated classes. Consequently, for any given real-life application of an image classification model, less severe mistakes are preferred over severe ones.

Semantic similarity can also be expressed in terms of a hierarchy that encodes the semantic relationship between classes. For example, a *German Shepherd* is a *dog*, which is an *animal*, which in turn is a *living thing*. Such a hierarchy allows us to assess how semantically similar two classes are. A *Horse* is somewhat related to a *German Shepherd*, as they are both *animals*, while an *airplane* is a distant concept, sharing none of the hierarchy with a *German Shepherd*. This sort of semantic relationship, i.e., that both *German Shepherds* and *Italian Greyhounds* are *dogs*, is often represented as a semantic tree. These trees encode hierarchical relationships, typically representing an "IS-A" relationship between parent and child nodes.

Hierarchical classification is a methodology that accounts for semantic similarity between classes using such a hierarchy. The task is not only to predict a single class but rather an entire hierarchy. In addition to potentially improving overall accuracy, a key objective is to limit misclassifications to semantically related classes. This approach is somewhat inspired by human cognition; while humans may confuse two dog breeds, they can typically discern that the entity

in question is a dog.

For hierarchical classification, a ground truth semantic hierarchy is needed. This makes the ImageNet-1k data set (Russakovsky et al., 2015) a sensible choice, as it is a large-scale image data set with classes that align with the WordNet hierarchy (Miller, 1995). This allows for creating a tree structure of "IS-A" relationships using different higher-level semantic concepts. Generally, there are two approaches to hierarchical classification (Bertinetto et al., 2020). It is possible to use a specialized architecture or a hierarchical loss function. Hierarchical loss functions aim to penalize predictions with a greater hierarchical distance to the true label more severely (Bertinetto et al., 2020), thus learning to avoid more significant mispredictions. Hierarchical architectures incorporate the hierarchical structure of the classes themselves to teach the hierarchy without necessarily modifying the loss function.

Various architectures have been proposed for hierarchical classification, such as the works of Zhu and Bain (2017), Seo and shik Shin (2019), Kolisnik et al. (2021), or Bilal et al. (2017). Many of these architectures have only been tried on relatively small data sets, such as CIFAR-10 or 100. This thesis argues for the importance of further exploration of hierarchical architectures which work with large-scale data sets. Besides the obvious benefit of accuracy enhancement, several other aspects make a deeper exploration worthwhile. Most importantly, hierarchical architectures may provide a deeper understanding of how neural networks learn the semantic relationships of classes. This deeper semantic understanding may help to increase the robustness of the model, which in turn may help to make fewer and less severe misclassifications.

A potential use case of such robustness is open-set classification. In open-set classification, networks should be aware of their limitations, and when encountering unknown classes, they should label them as such. By training a model to have a better understanding of the semantic relationship between classes, they may be better able to reject such unknown classes. This is especially interesting as most open-set research has focused on smaller data-sets and not large-scale ones (Palechor et al., 2023).

Given these potential benefits, this thesis explores different hierarchical architectures for predicting hierarchical classes of samples from the ImageNet data set. Three main research questions guide this exploration.

> *RQ 1: Does a CNN outperform baseline models when predicting hierarchical classes jointly?*

This first research question explores whether a single multi-task model is better than separate models. The single multi-task model uses a shared backbone followed by separate classifiers for each hierarchical level, simultaneously predicting the classes of all levels. This model is compared to the performance of multiple models, each specialized for predictions of a single hierarchical level. This question's answer serves as a foundation for more advanced architectures and explores the impact of sharing a backbone structure. The single model with a shared backbone aims to use synergies between the tasks to increase performance. Building on these findings, further improvements to this model's architecture are explored. Concretely, it is explored whether information can be shared between the classifiers:

> *RQ 2: How can information be shared between hierarchical levels to boost performance and increase semantic understanding?*

In particular, two architectural modifications are explored:

> *RQ 2.1: Does sharing the probability distribution of a higher hierarchical level improve performance on a lower hierarchical level?*

> *RQ 2.2: Is it possible to create an attention mechanism using the weights of a higher hierarchical level to improve performance?*

The first of these two models remains relatively simple and engineers the features of the backbone so that the prediction of the previous hierarchical levels is included. The second model is more sophisticated and aims to improve performance by introducing an attention mechanism to share information between hierarchical levels. Attention mechanisms have shown great promise for computer vision tasks, such as the Squeeze-and Excitation Network (Hu et al., 2018) or Vision Transformer (Dosovitskiy et al., 2020). This work introduces an attention mechanism that highlights relevant features between hierarchical levels without increasing the number of parameters by transforming deep features based on the predicted higher-level classes. These research questions are answered through several experiments that compare the different architectures with each other and measure their performance on each hierarchical level. In conjunction with the previous research questions, a final research question explores how well these models can distinguish known from unknown classes:

RQ 3: How do the resulting networks perform on open-sets?

For this, classes that are not included in the learned hierarchy, and are thus unknown, are shown to a selection of well-performing models.

Chapter 2

Related Work

This chapter discusses the related work of hierarchical classification, open-set recognition, and attention mechanisms in CNNs.

2.1 Hierarchical Classification

In traditional image classification, images are categorized into a fixed set of classes. This is often referred to as flat classification Yan et al. (2015). Each of these classes is independent of one another. For this task, convolutional networks with a deep architecture have proven exceptionally well suited (He et al., 2016). However, flat image classification disregards the semantic relationship between classes. Humans naturally understand that the semantic distance between a *German sherpard* and an *Italian greyhound* is much smaller than between a *German sherpard* and an *Airplane*. This limitation is addressed by hierarchical classification, which organizes classes into a tree-like structure. Rather than predicting a single class, the hierarchical structure of a given class is predicted.

Generally, models integrate hierarchical information through either a loss function or the architecture. Hierarchical loss aims to penalize predictions for classes with a large hierarchical distance to the true label more than predictions of closely related classes (Bertinetto et al., 2020). Several approaches have been taken to integrate such a loss function into CNN. Examples for this are the works of Wu et al. (2016), Bilal et al. (2017), or Bertinetto et al. (2020). Such approaches often directly aim to avoid severe misclassifications by highly penalizing mistakes that occur high up in the hierarchy. Hierarchical architectures do not necessarily change the loss function but rather aim to adapt the network architecture at a structural level so that hierarchical information is considered during training. A common approach is to assign superclasses to higher layers of the network and to determine more granular classes at the lower layers (Bertinetto et al., 2020). This is based on findings by Zeiler and Fergus (2014), where the analysis of a CNN showed that its earlier layers learn low-level features while the latter layers learn high-level features.

Hierarchical loss and architecture can also be combined (Bilal et al., 2017). Both approaches – and the combination thereof – offer potential solutions for hierarchical classification. Still, the focus of this work lies in exploring how the architecture can be explicitly designed to predict a hierarchy of a given class. This directly leads to a key problem when predicting a hierarchy. That is, how the hierarchy is determined. The previous literature again followed two distinct approaches to address this issue. Either a (manually) predefined or automatically clustered method is used. Furthermore, a distinction between different hierarchies can be made: "Has-A" hierarchies describe a relationship where one entity (physically) contains another entity. An "IS-A" hierarchy describes a parent-child relationship where a parent node semantically contains a child node (Guo et al., 2018). This work aims to explore the nature of "IS-A" hierarchies, which can also

be described as superclass and subclass relationships. There have been various architectures that tackle hierarchical classification.

Yan et al. (2015) propose a generic hierarchical architecture called *Hierarchical Deep Convolutional Network* (HD-CNN). It captures features at different layers to predict both a coarse and fine class. Based on the (overlapping) coarse class, which is predicted with features from earlier layers, different fine classifiers then make a final prediction. The hierarchy is created from the confusion matrix created by a dedicated classifier from the training data. This approach creates coarse classes from classes that are confused with each other by the dedicated classifier. This is an automated approach that helps to increase performance, as it allows more granular distinctions of classes that are related to each other. Still, it should be noted that this does not imply a semantic relationship between these classes, as misclassification likely happens due to visual similarity between classes rather than due to their semantic relationship.

Zhu and Bain (2017) expands on this work by allowing it to scale to multiple coarse categories. The resulting architecture is called *Branch Convolutional Neural Network*. (B-CNN). Instead of being limited by a single coarse category, it can predict multiple coarse categories. At multiple levels, deep features are used for predictions. Each of these categories has its own classifier, which can be seen as a branch that spreads out from the shared layers. The results of all branches are then again used in a final prediction for the most granular class. They use VGG16 (Simonyan and Zisserman, 2015) as a backbone and propose a custom training strategy. One of the benefits of predicting coarse classes in higher layers is to combat the problem of vanishing gradients. In their solution, they either manually create a semantic hierarchy with a limited number of classes and levels or use predefined hierarchies, such as the superclasses from the CIFAR-100 data set (Krizhevsky, 2009). Note that this work has never been published in a peer-reviewed journal. However, as other related works which have been published reference this work, it is added for completeness' sake.

Similarly, Inoue et al. (2020) further modifies the idea of B-CNN with two architectures: Concat-net and Add-net. This architecture again predicts one or more coarse classes and the final fine class using the deep features of multiple layers. The main difference is that the output of the dense layer at each hierarchical level is either concatenated or added to the next dense layer. The core idea is to further enhance the ability to share information between hierarchical layers. The hierarchies were again either manually constructed or taken from the CIFAR-100 data set.

Seo and shik Shin (2019) aimed to classify images from the Fashion-MNIST data set with a VGGNet backbone. They manually created a hierarchy with two coarse classes. They again use deep features from different blocks to create predictions of the coarse classes. Kolisnik et al. (2021) provide a further extension of the B-CNN architecture. Its predictions of the coarse classes are multiplied by a conditional probability weight matrix in order to obtain the lower-level predictions.

Bilal et al. (2017) combine both a hierarchical loss function with a hierarchical architecture. They used AlexNet (Krizhevsky et al., 2017) as a backbone. To this backbone, multiple branches are added during training to impose the hierarchical structure. During testing, this hierarchical structure is removed. The authors propose that understanding hierarchical learning behavior can improve CNN design and suggest designing hierarchy-aware CNNs to accelerate model convergence and alleviate overfitting.

Other approaches derivate from taking deep features from upper layers to predict coarse classes. Roy et al. (2020) directly recreate a tree-like structure with separate classifiers at each node. Each node classifies the image into one of its children. Unlike the other described approaches, Guo et al. (2018) focus on "Has-A" relationships. They combine CNNs with recurrent neural networks (RNNs) to predict the hierarchy of a given class as a label path. This approaches the problem of predicting a hierarchy path as predicting sequential data.

Note that hierarchies have also been used to improve classification while not being described

as hierarchical classification. One example is YOLO9000 (Redmon and Farhadi, 2017), which uses the WordNet hierarchies to improve on Redmon et al. (2016) work. The network is trained with multiple labels from the WordNet hierarchy. In addition to the 1000 ImageNet classes, 369 intermediate classes are predicted in order to learn a hierarchical structure. Redmon and Farhadi (2017) found that this results in slightly lower accuracy but much greater performance when confronted with unknown objects.

2.2 Open-Set Recognition

Most approaches described in Section 2.1 assume a closed-world setting when evaluating the proposed models. One notable exception is YOLO-9000, which is trained on one data set, and then its performance is additionally evaluated on a second data set. This approach goes into the domain of open-set recognition. Some research has even explicitly focused on combining open-set classification with hierarchical classification (Lee et al., 2018). Still, this topic remains largely unexplored. A key challenge of any deployed deep neural network is that it may encounter classes during testing that it has never encountered during training. If a neural network encounters a sample of an unknown class, it should be able to recognize that it is not any of the learned classes. There have been various methods to approach this issue, such as the works by Bendale and Boult (2016), Ge et al. (2017), Dhamija et al. (2018), or Chen et al. (2020). However, many of these approaches only employ small-scale data sets – the work of Bendale and Boult (2016) is a notable exception. Recently, there has been an effort to develop standardized evaluation metrics that can be used to compare different open-set algorithms (Palechor et al., 2023). Such an evaluation approach can also be applied to algorithms that have not been *explictly* developed to handle such unknown classes. Many algorithms have been trained with *known unknown* or *negative* samples (Bendale and Boult, 2016), Ge et al. (2017). These samples are given to an algorithm during training with a label that indicates that this sample should be classified as an unknown class to teach a model the difference between known and unknown samples. Still, ideally, even an algorithm unaware of any negative samples should be able to reject unknown classes. A simple approach is to threshold the maximum class probability (Palechor et al., 2023). The hope is if a model receives an unknown input, it would not confidently predict a single class but rather that the probability would be distributed across all classes. While this approach is not the most effective, it can easily be implemented to compare how networks not specially trained for open-set classification behave when confronted with unknown samples. Furthermore, there are certainly use-cases where such thresholding is effective, as shown by Hendrycks and Gimpel (2016) and Vaze et al. (2021). This may be highly interesting for hierarchical classifiers, especially to see whether they can reject samples outside the known hierarchy.

2.3 Attention in CNNs

Attention mechanisms have been becoming increasingly popular in deep learning techniques. Especially in large language models (LLMs) attention mechanisms have been successfully integrated into powerful models such as transformers (Vaswani et al., 2017). Similar mechanisms have also been applied to computer vision tasks to enhance – or even replace – traditional CNN architectures. In traditional CNNs, the same convolutional filter is applied to the entire input image. Furthermore, they are often only able to learn localized relationships, such as the relation between a pixel and its neighbors. Attention mechanisms aim to overcome this by learning which features are relevant to a given task and then somehow incorporating this information.

One example of a CNN that uses an attention mechanism is the non-local neural network (Wang et al., 2018). It uses a spatial attention mechanism to capture the long-distance relationship between pixels. This is achieved via non-local operations. The operations compute a weighted sum of all the input features rather than just localized features. The main application, however, is video classification and not image classification.

Hu et al. (2018) introduce the squeeze and excitation (SE) network with the main purpose being image classification. While spatial attention focuses on specific image regions, channel attention emphasizes important channels within feature maps. Using multiple SE blocks, the SE network recalibrates the channel-wise features by modeling interdependencies between channels. The network achieves through a two-step process: squeeze and excitation. In the squeeze step, the spatial dimensions of the input feature map are squeezed into a number of channels through global average pooling. During the excitation step, channel-wise dependencies are captured and then used to recalibrate the original feature map to highlight the relevant information.

Woo et al. (2018) propose an architecture that combines both spatial and channel attention into a single block. First, a 1-D channel attention map is calculated, and then a 2-D spatial attention map. These are sequentially applied to an intermediate feature map via element-wise multiplication.

These approaches depict rather general attention mechanisms, which can be added to most network architectures. There have also been other, more task-specific attention mechanisms, such as for facial expression recognition (Li et al., 2020), or Image denoising (Tian et al., 2020).

There even have been attention-based architectures that do not use CNNs at all for image classification. These approaches generally rely on self-attention instead of convolutional layers. The most famous of these architectures is the Vision Transformer (ViT) proposed by Dosovitskiy et al. (2020). ViT interprets an image as a series of patches and processes this sequence just as a language model would. This approach proved very successful in image classification tasks, one of the main advantages being lower computational requirements to achieve high accuracy scores. Recently, there have been attempts to integrate self-attention with CNNs (Pan et al., 2022), outperforming comparable models that require the same amount of processing power.

Chapter 3

Background

This chapter aims to provide a comprehensive overview of convolutional neural networks and hierarchical classification. This includes their theoretical foundations and key concepts to provide a solid foundation for the subsequent chapters of this thesis.

3.1 Supervised Learning

The most common setting for image classification tasks is supervised learning (LeCun et al., 2015). Assume one wants to use machine learning to classify an image into whether it is a dog or a cat. First, a large data set of images containing dogs or cats is collected and manually labeled according to their category. A given neural network is shown a set of images during training. Based on the input image, the network produces an output – in this case, a prediction on the class of the image. Such a neural network consists of multiple layers, each with its own set of learnable parameters. The goal is to adjust these parameters in such a manner that the network can be given a previously unseen input image and produce a prediction that matches the labeled category (either a cat or a dog) of the image.

A simple example of this is a single-layer neural network. In mathematical terms, this can be depicted as a formula:

$$\vec{y} = g(\mathbf{W}\vec{x} + \vec{b}) \tag{3.1}$$

Here, $\mathbf{W}$ is the weight matrix of the neural network, $\vec{x}$ is the input image, $\vec{b}$ is a bias term, g is an activation function, and $\vec{y}$ is the output. Each row of $\mathbf{W}$ corresponds to the weights connecting one output neuron to all neurons in the previous layer. The goal is to iteratively adjust the weights so that the network makes correct predictions. This is achieved through a process called back-propagation. Rumelhart et al. (1986) introduced this learning procedure with the key idea of adjusting parameters, i.e., the weights, based on an error function. The error function, commonly called the loss function, is essential for enabling neural networks to learn. It expresses the distance between the predicted outcome and the ground truth outcome, thus minimizing the loss function results in the objective being reached. Based on this function, one can calculate the gradient - the negative of which describes the direction of the steepest descent. This can be seen as the path toward the minimum of the loss function. The gradient of the loss function $\mathcal{L}(\mathbf{W})$ with respect to the weights can be expressed as:

$$\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}) = \frac{\partial \mathcal{L}(\mathbf{W})}{\partial \mathbf{W}} \tag{3.2}$$

Adjusting the weights using their gradients is an iterative process. In practice, most commonly, a procedure called stochastic gradient descent (SGD) is used (LeCun et al., 2015):

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}_t) \tag{3.3}$$

$\mathbf{W}_t$ is the weight matrix at the current iteration, which gets updated based on the gradient $\nabla_{\mathbf{W}}\mathcal{L}(\mathbf{W}_t)$ of the loss function with respect to the weight matrix at the current iteration multiplied with the learning rate η. The learning rate η controls how much the weights get updated: If it is too large, SGD may overshoot the minimum and fail to converge. If it is too small, convergence may take a long time. SGD often uses many small batches of examples to approximate the average gradient over the whole data set, hence the name stochastic.

Which loss function should be minimized depends on the problem. The above-described scenario of classifying dogs and cats is a binary problem: Any given sample is either a cat or a dog. In most real-world scenarios, there are many more than two classes. This setting is referred to as categorical classification. For categorical classification, the most common loss function is categorical cross-entropy:

$$\mathcal{L}_{CCE} = -\frac{1}{N}\sum_{n=1}^{N} \vec{t}_n \odot \log \vec{y}_n \tag{3.4}$$

This loss function represents a similarity measurement between the categorical distribution $\vec{y}$ and the actual one-hot encoded target vector $\vec{t}_n$ for all samples $\vec{x}_n$. The symbol $\odot$ represents element-wise multiplication. The formula calculates the negative logarithm of the predicted probability for the true class label.

This means that for each sample, a discrete probability distribution is predicted. Based on this probability distribution, the loss function calculates the distance between the predicted categorical distribution and the target vector. Neural networks often consist of multiple layers as introduced in (3.1). Such a layer is called fully connected (FC) since the weight matrix connects each output element with all input neurons. The softmax function is commonly applied to a categorical distribution to obtain a normalized output vector that sums up to 1. This allows for an easier interpretation of the output:

$$\text{softmax}(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{3.5}$$

Here $\vec{z}$ are the logits of size C, which is the total number of classes to be predicted. The softmax function transforms these logits into probabilities by exponentiating each element (to ensure non-negativity) and normalizing the result.

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are specialized neural networks that employ a mathematical operation called convolution. Goodfellow et al. (2016) describes them as neural networks that use convolution instead of general matrix multiplication in at least one of their layers. 2D convolution is commonly used to process data with a grid-like topology, e.g., image data. Traditional neural networks apply a series of fully-connected layers, meaning every input unit interacts with every output unit. Convolutional layers typically have sparse interactions, which can be achieved through the use of a kernel that is smaller than the input. This process allows the processing of spatial relations and, thus, the extraction of meaningful features. Images have multiple channels (e.g, RGB), so an image can be seen as a 3D input $X \in \mathbb{R}^{Cin \times H \times W}$, where C_{in} represents the number of channels, H the height, and W the width of the image. As such, convolutional neural network iterate over multiple channels. Formally, a convolution operation, adapted from Goodfellow et al. (2016), is described as:

$$\mathbf{Y}_{i,j,k} = \sum_{c}^{C_{in}} \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} \mathbf{X}_{i+m,j+n,c}\mathbf{K}_{m+M/2,n+N/2,c,k} \tag{3.6}$$

The four-dimensional kernel K slides over the image X producing the output feature map Y. Besides the two spatial dimensions, the channels of the input C_{in} are considered. $\mathbf{Y}_{i,j,k}$ describes the output at the location i, j for each output channel k in C_{out}. At each output index, the value is the sum of the element-wise multiplication between the kernel K and a sub-region of the input image X centered at that index. The kernel limits are M and N, which are usually odd. Since indexing is used, $M/2$ and $N/2$ are integer divisions, so m and n iterate over all indices of the kernel, with the center of the kernel being at index $0, 0$.

The small size of the kernel also means that such a layer uses a smaller number of parameters. While CNNs have been around for over 30 years (LeCun et al., 1989), there have been significant improvements in performance in various computer vision tasks due to the emergence of deeper architectures and increased computing resources. In deeper architectures, a large number of layers are stacked behind each other. A convolutional network consists of multiple simple layers. These layers can be categorized into three stages (Goodfellow et al., 2016). In the first stage, the layer performs multiple convolutions introduced in (3.6) to produce a set of linear activations. In the next stage, the activations are run through a non-linear activation function. The third stage is a pooling function which modifies the output further. Pooling functions modify the output at a certain location to be a summary statistic of the nearby outputs (Goodfellow et al., 2016). This helps the net to be invariant to small transformations in the input. These three types of layers are repeated multiple times; the more layers are used, the deeper the architecture. Finally, one or multiple FC layers are used for the final classification. One of the milestones for deep CNNs was AlexNet, introduced by Krizhevsky et al. (2017). This network had a depth of 8 layers, which was considerably deeper than most architecture at that time. Following the success of AlexNet, other deep architectures appeared, such as VGGNet by Simonyan and Zisserman (2015) with a depth of 19 layers. However, these deeper architectures also came with their challenges. Deeper network architectures not only lead to saturation of accuracy but rather a degradation in performance (He et al., 2016).

3.3 ResNet Architecture

The ResNet (Residual Network) is proposed by He et al. (2016) to make increasingly deep architectures possible through residual connections. A common problem with deep networks is vanishing gradients: As the gradient is back-propagated through many layers, the gradient signal becomes increasingly weak. As the weights are adjusted based on the gradient, a network can no longer learn once the gradient has vanished. This results in a situation where the network can no longer effectively update its weights (Ioffe and Szegedy, 2015). This issue is commonly addressed using non-linear activation functions such as ReLU (Nair and Hinton, 2010) or batch normalization (Ioffe and Szegedy, 2015). While vanishing gradients are a problem, He et al. (2016) address a slightly different issue. Degradation in the performance of deep CNNs is *unlikely* caused by vanishing gradients. The authors verified that neither forward-propagated signals nor backward-propagated gradients vanish. Rather, the main issue can be seen in the difficulty of approximating identity mappings by multiple nonlinear layers, leading to decreased performance with increasing depth. He et al. (2016) aim to solve this problem with deep residual learning. The core idea of ResNet is to approximate residual mappings rather than directly learning a mapping. Every few blocks, a residual building block is inserted, formally defined as:

$$\vec{y} = \mathcal{F}(\vec{x}, \{\mathbf{W}_i\}) + \vec{x} \tag{3.7}$$

Here $\vec{x}$ and $\vec{y}$ are the input and output vectors of the considered layers. The function $\mathcal{F}$ represents the residual mapping to be approximated. The operation $\mathcal{F} + \vec{x}$ is a residual connection,

Table 3.1: Architecture of ResNet-50 adapted from He et al. (2016).

Type/Name	Output Size	Details
Input	224×224	
Conv1	112×112	7x7, 64, stride 2
Conv2_x	$56 \times 56 \times 256$	3x3 max pool, stride 2
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
Conv3_x	$28 \times 28 \times 512$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
Conv4_x	$14 \times 14 \times 1024$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
Conv5_x	$7 \times 7 \times 2048$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
FC	1×1000	average pool, 1000-d FC, softmax

where the input of the block is added element-wise to the output of the residual mapping. Essentially, this network learns the *difference* between the desired output and the input instead of directly learning the desired output. This simplifies what each layer has to learn with the above-described residual connection. The result is an architecture that increases its performance with depth while the residual connections introduce minimal additional computational requirements. There are various versions and modifications to ResNet architectures. In the original paper He et al. (2016) propose 18, 34, 50, 101, and 152-layer architectures. Based on their finding, residual connections are applied after two layers in the ResNet-18 and 34 architectures and after three in the other architectures.

The performance and resource requirements both increase with the number of layers used. ResNet-50 is a compromise between good performance and comparatively short training time. The layers are listed in Table 3.1. Note that this depiction is incomplete, as it lacks the depiction of the shortcut connections and the ReLU activation functions. ResNet-50 consists of 18 blocks. In total, there are 48 convolutional layers, one MaxPool layer, and one AveragePool layer. Finally, a single FC layer is used to predict logits. The softmax function shown in (3.5) converts the raw logits into the normalized probability distribution over all classes.

The convolutional layers provide a 2'048 dimensional deep feature map, which is then classified by an FC layer. The FC layer depends on the number of classes to be classified, which is 1'000 for the ImageNet-1k data set (Russakovsky et al., 2015). These factors make ResNet a strong backbone choice for training from scratch.

Chapter 4

Data

Large-scale data sets of annotated images are essential for deep learning techniques in image classification, as it enables the training of CNNs on millions of pictures belonging to thousands of classes. ImageNet (Deng et al., 2009) is one such data set that has been instrumental in advancing the field of computer vision.

4.1 WordNet

The annotated classes of ImageNet are structured according to the WordNet lexical database (Miller, 1995). WordNet is, as the name suggests, a network of words with their semantic relations. While various semantic relationships exist in WordNet, ImageNet is only structured according to the concept of synonymy. Each meaningful concept is described by a set of synonyms (synset), i.e., different words that describe the same concept. However, as proposed by Deng et al. (2009), the main asset of using WordNet is the ontology of its concepts. This can be achieved via a second semantic concept in WordNet: hyponymy. This concept describes a super-subordinate relation, i.e., an "IS-A", relationship between synsets. An important property of hyponymy relation is transitivity (Princeton University, 2010). For example, as *dachshund* is a hyponym of *dog* and *dog* is a hyponym of *mammal*, transitivity means that *dachshund* is also a hyponym of *mammal*. As ImageNet is structured according to WordNet synsets, one can create a hierarchical tree using WordNet hyponyms.

4.2 ImageNet

At the time of release, 5247 categories – each category corresponding to a WordNet synset – were represented in ImageNet (Deng et al., 2009) containing 600 images on average. This made it one of the largest labeled data sets available at the point of its release, and it has been further extended. According to the ImageNet website (Yang et al., 2021), over 14 million images of 21'841 synset have been collected. It contains categories of a diverse nature, covering animals, people, other organisms, and inanimate objects. The images are also diverse in the poses, cluttering, and background of the images.

A subset of this larger database has been introduced by Russakovsky et al. (2015) for the ImageNet Large Scale Visual Recognition Challenges (ILSVRCs). The most recent subset (ILSVRC 2012) contains 1'000 classes and is often called ImageNet-1k. This publicly available subset of the ImageNet data set contains 1'281'167 training images, 50'000 validation images, and 100'000 test

images, and all 1'000 classes are manually annotated. Note that the test images do not have publicly available labels. The 1'000 classes were chosen through a combination of automatic heuristics and manual post-processing. It is also ensured that each class contains a suitable number of images. The training set contains 732 to 1300 images per class, while the validation set always contains 50 and the test set 100 images (Russakovsky et al., 2015).

Due to its size and availability, ImageNet-1k is a popular benchmark that allows the comparison of different architectures. Milestone architectures of CNNs, such as AlexNet (Krizhevsky et al., 2017), have been proposed during the ILSVRC. In these challenges, held from 2010 to 2017, various tasks were to be solved on the ImageNet-1k data set. These tasks include image classification, single-object localization, and object detection. From ILSVRC-2012 onwards, the ImageNet-1k data set remains unchanged and is used in this thesis.

The images of ImageNet-1k consist of objects and living beings. The 1'000 classes were selected so that no overlap between synsets exists Russakovsky et al. (2015). This means that no ImageNet classes have a super-subordinate relationship with each other. However, these classes share common superordinates in the WordNet structure. It should also be noted that the ImageNet-1k classes are not necessarily end nodes (or leaves) in the WordNet structure. So any ImageNet class may have further, more granular classes in the WordNet hierarchy. The WordNet hierarchy is structured as a directed graph, so some synsets have multiple ancestors. This may lead to issues as some images contain multiple objects.

Further issues of the ImageNet-1k data set include misclassifications of wildlife (Luccioni and Rolnick, 2023) and general low-quality labels. Northcutt et al. (2021) claims that up to 6% of the ImageNet validation set contain label errors. While the *intention* of the ImageNet-1k authors (Russakovsky et al., 2015) was to create non-overlapping synsets, some overlapping classes exist (Luccioni and Rolnick, 2023): For example, the class *meerkat* refers to a type of *mongoose*, despite *mongoose* being a different ImageNet class. Despite these issues, the ImageNet-1k enjoys immense popularity as a benchmark data set.

4.3 Superclasses for WordNet Hierarchy

Combining the class labels from ImageNet-1k with WordNet hyponyms allows for the creation of the WordNet hierarchy. Although this hierarchy is extensive, it is not well suited for hierarchical classification without some modifications. One of the main reasons for this is that the WordNet hierarchy is a directed graph and not a tree structure. Essentially, this means that any given node in that structure may have multiple ancestors. Building the hierarchy results in a graph with 1'856 nodes. Only 37 of these nodes have multiple parents. However, if a node higher up in the hierarchy has multiple ancestors, this ambiguity propagates down.

Furthermore, the depth inside the hierarchical tree varies greatly between different objects. The node depth, i.e., the number of nodes between a given node and the root, varies between 5 and 18, with a mean of 11.3. Classes of animals tend to be embedded much deeper in the tree. This is mostly because, for animals, biological taxonomy is used in WordNet. For example, the distance in the tree between a *killer whale* and a *great white shark* is 14, i.e., 14 nodes must be traversed to get from one node to the other. These 14 nodes represent the distance in the biological taxonomy of these two animals. Meanwhile, the distance between a *baseball player* and a *cliff* is only 7. This difference in tree depth already highlights a key challenge when creating hierarchies for real-world entities. An arbitrary number of hyponyms could be found for any word, especially if no clear scientific taxonomy is used. This means that any hierarchical classification is inherently dependent on the ontology of the hierarchy.

To get a hierarchy that a neural network for hierarchical classification can easily handle, this graph must be converted into a tree, as an unambiguous hierarchy is needed. Furthermore, the

Algorithm 1 ROBUSTNESS' SUPERCLASSES. *This algorithm takes the entire WordNet/ImageNet tree (sorted by the number of descendants of each node) and creates a mapping of $n\_superclasses$. Only descendants of $ancestor\_wnid$ are considered in this mapping – "wnid" standing for the WordNet ID for a synset.*

procedure GET_SUPERCLASSES($n\_superclasses$, $ancestor\_wnid$)
  $superclass\_info \leftarrow$ empty list
  for $node$ in $sorted\_tree$ **do**
    if length of $superclass\_info$ equals $n\_superclasses$ **then**
      break
    end if
    if $node$ is a descendant of $ancestor\_wnid$ **then**
      append ($node$, number of descendants of node) to $superclass\_info$
      $skip\_node \leftarrow$ False
      for $super\_node$ in $superclass\_info$ **do**
        if $node$ is a descendant of $super\_node$ **then**
          remove $super\_node$ from $superclass\_info$
        end if
      end for
    end if
  end for
  $class\_ranges \leftarrow$ get subclasses for each superclass in $superclass\_info$
  $superclass\_wnid, label\_map \leftarrow$ get corresponding wnid and label for $superclass\_info$
  return $superclass\_wnid$, $class\_ranges$, $label\_map$
end procedure

hierarchy should have equal depth for all child nodes for a simple architecture. A reason for this is also that in classification tasks, a fixed number of classes is usually required, as an FC classification layer predicts a fixed number of classes. If an arbitrarily long sequence should be predicted, the problem must be formulated as a sequence prediction task. While certainly possible, this would represent an orthogonal approach to the one taken in this thesis. Instead, in this thesis, the WordNet tree is taken and transformed into a simpler hierarchy containing higher- and lower-level concepts. Many related works use a three-level hierarchy like Zhu and Bain (2017), Seo and shik Shin (2019), or Kolisnik et al. (2021). Theoretically, it is possible to construct an arbitrarily deep hierarchy. However, the focus of this thesis lies not on the construction of the hierarchy but rather on the subsequent classification task. Under the consideration that any simplified hierarchy simply does not represent reality, any created hierarchy is arbitrary to a certain extent. As such, the constructed hierarchy can be seen as an extension of the data set: Various arbitrary decisions must be made. However, the same is true for any given data set.

The entire abstraction into a simplified hierarchy is handled by the `robustness` python package (Engstrom et al., 2019). First, this algorithm reads the WordNet graph and constructs a child-parent tree from the directed graph from provided files. The WordNet data is provided on the ImageNet website.[1] This file contains "IS-A" relationships between different words. As some nodes have multiple parents, it resolves ambiguity by only taking the parent that appears first in this file, and parents that appear later are not considered. Other, more sophisticated approaches are possible, such as adding new nodes to existing parents rather than adding new parents, as done by Redmon and Farhadi (2017). Still, no matter how the tree is constructed, it will contain semantic information from the WordNet hierarchy. It is also important that the tree is constructed

[1] https://image-net.org/data/wordnet.is_a.txt

deterministically for reproducibility.

From this ImageNet/WordNet tree, one can create the simplified ground-truth hierarchy used for classification. To create higher and lower-level concepts, a simple splitting algorithm implemented by `robustness` (Engstrom et al., 2019) can be used. The implemented algorithm has a relatively simple approach depicted in Algorithm 1. A parent node $ancestor\_wnid$ and the desired number of superclasses $n\_superclasses$ are given as an argument. The algorithm then iterates over the descendants of the provided parent node and selects the nodes with the most descendants, i.e., ImageNet-1k classes. This superclass is then added to a list containing the superclasses. Each time a new superclass is added, the algorithm checks if it is a descendant of an existing superclass. If so, the ancestor superclass is removed from the list, and the new superclass is added. Otherwise, the superclass is directly added. As a parent class can be given as an argument, it is possible to recursively do such a split to get superclasses on different hierarchical levels. However, it does not result in a solution that contains all ImageNet classes. The number of ImageNet classes included depends on how many levels the hierarchy should have and the number of classes at each hierarchical level. While this is satisfactory for preliminary approaches, a complete way of creating such a hierarchy may be desirable. An outline for creating a hierarchy that contains *all* ImageNet classes can be found in Appendix A.1.

To create a ground-truth hierarchy from this algorithm, one must decide on the depth and breadth of the resulting hierarchy. In this thesis, a three-level hierarchy is used, just as in many related approaches. These three levels are referred to – from top to bottom – as *superclasses*, *subclasses*, and *target classes*. The ImageNet data set consists of two main concepts: *living things* and *artifacts*. This provides a simple answer for the number of superclasses. In total, 928 of the 1'000 ImageNet are contained in these two superclasses: 521 artifacts and 407 living things. Here, the input for Algorithm 1 is the root node as $ancestor\_wnid$ and 2 for $n\_superclasses$. Selecting the number of subclasses is more challenging. In this thesis, seven subclasses for the artifact superclass were chosen, and five subclasses for the living things superclass. This choice is fairly arbitrary and aims to allow for a trade-off between classes that are not too general and not too specific. These subclasses are again generated with Algorithm 1, where the WordNet ID of the two superclasses is used as an $ancestor\_wnid$ and the number of desired subclasses for $n\_superclasses$. Using this split, one gets the hierarchy depicted in Figure 4.1. In total, 660 ImageNet classes make up the lowest hierarchy level. These classes are grouped into 12 subclasses: The artifact superclass consists of the classes *device*, *structure*, *clothing*, *wheeled vehicle*, *covering*, *equipment*, and *implement*. The living thing superclass has the subclasses *placental mammal*, *hunting dog*, *arthropod*, *bird*, and *reptile*.

Note that there are some issues with the constructed hierarchy. First and foremost, there is both a subclass *placental mammal* and a *hunting dog* one, even though dogs are mammals. This is due to the ambiguous ancestry of the *dog* node in the ImageNet/WordNet tree. This node is a descendant of both the *placental mammal* and the *domestic animal* nodes. However, during the tree construction, the *domestic animal* node is picked as the ancestor node. A further (potential issue) is that only the 61 descendants of the *hunting dog* node are considered, even though the *dog* node has 116 descendants. This is because Algorithm 1 splits up big superclasses in favor of smaller ones if the number of superclasses has not been met. As such, the *dog* node is replaced by the smaller *hunting dog* node.

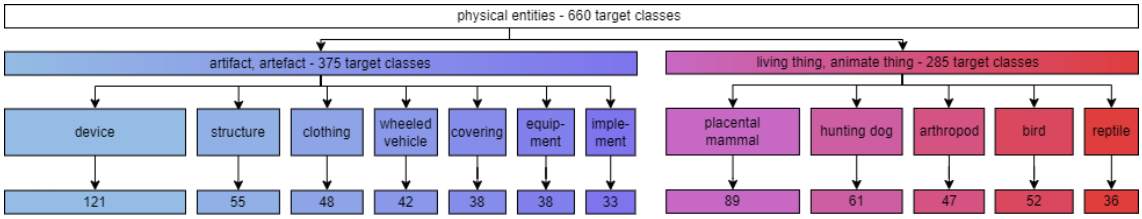


Figure 4.1: HIERARCHY FOR LEARNING. *This figure depicts the resulting classes on the three hierarchical layers. The lowest layers - the target classes - are depicted as the number of target classes of a given subclass.*

Chapter 5

Approach

To explore the learning semantics of hierarchical classification, a common learning setting must be established. The task to be solved is the hierarchical classification of the ImageNet-1k (ILSVRC2012) Imagenet data set.[1] Hierarchical classification is used to teach the neural network semantic relationship between different ImageNet classes. The overarching goal is to find a neural network architecture that can distinguish similar classes, i.e., semantically related ones, from dissimilar ones. This chapter provides an overview of the approach taken in this thesis.

5.1 Architecture for Hierarchical Classification

To perform hierarchical classification, it is important to choose an appropriate architecture that can handle the hierarchical structure of the classes. It is certainly possible to use a naive approach and predict all hierarchical levels independently using multiple networks. This comes with high computational costs and does not result in a single model which has learned the underlying hierarchy. Instead, to explore the learning semantics of neural networks, the approach is to predict all hierarchical layers with a single network jointly. Besides resulting in more efficient training - as only one network needs to be trained - joint classification may even lead to increased performance. This is especially true if there are correlations between the labels to be predicted (Zha et al., 2008). For hierarchical classification, this correlation is especially strong: If a given target class has the label *foxhound*, it always also has the labels *hunting dog* and *animate thing*. A multi-objective network, thus, is likely to outperform single-task networks at higher hierarchical levels. Still, separately trained models may enjoy advantages at lower hierarchical levels.

5.1.1 Baseline

To assess whether multi-task networks are an approach worth exploring, the alternative option of multiple single-task networks must be considered. ResNet-50 is used as a backbone, as the goal is not the highest possible accuracy on the target classes, and ResNet-50 allows for fast training while achieving decent performance. The three separate networks are called the *Baseline* model. It serves as a baseline for how well hierarchical classification can be solved by an ensemble of models that do not share any information between them. The model consists of three ResNet-50 networks, each solving the single-objective classification task for a given hierarchical level, depicted in Figure 5.1(a). Each hierarchical level is predicted with a single ResNet-50 network, which takes an image as input and predicts the C_b classes for each level. Such an approach has the

[1] https://www.image-net.org/challenges/LSVRC/2012/

obvious downside of requiring the training of an entire network per hierarchical level, leading to high computational costs and many parameters.

5.1.2 Backbone Share Net

The more natural approach is to predict all hierarchical layers with a single network jointly. A popular approach for single-model solutions is to use features from earlier layers to classify classes higher up in the hierarchy. Features from later layers are then used to classify more granular classes. Such an approach has been proposed in the works of Zhu and Bain (2017), Inoue et al. (2020), or Seo and shik Shin (2019). This is mostly inspired by the findings of Zeiler and Fergus (2014) who use Deconvolutional Networks to visualize the activations of CNNs. The authors found that CNNs learn to recognize a hierarchy of image features. Earlier layers learn low-level features, such as edges or corners, while later layers learn high-level features, such as textures or entire objects. The approach taken in this thesis differs from that: Rather than using feature maps from different layers, a shared backbone is used to make hierarchical predictions. Deep features refer to the output of a given layer in a CNN – in this case, the output of the last layer before the FC layer of the ResNet-50 backbone. This is more similar to approaches such as those of Yan et al. (2015) or Li et al. (2021). The reasoning behind this is that lower-level features do not necessarily define higher semantical concepts. As such, it is reasonable to assume that all three hierarchical levels may use information encoded in the same deep features.

From the shared backbone, the deep features are taken and given to multiple branches. Each of these branches predicts a hierarchical level. This provides a simple architecture for hierarchical classification. To further explore the semantics of hierarchical learning, several different approaches for how these branches work are considered. The shared backbone is a ResNet-50 backbone where the FC-layer is removed to produce a feature $\vec{\varphi}$. The ResNet50's architecture has been described in Section 3.3 in detail. Since the final layer is removed and this backbone only consists of 49 layers, the backbone is referred to as ResNet-49. The deep features $\vec{\varphi}$ are a 2048-dimensional vector. In this thesis, this approach is referred to as the *Backbone Share Net* (BSN) depicted in Figure 5.1(b).

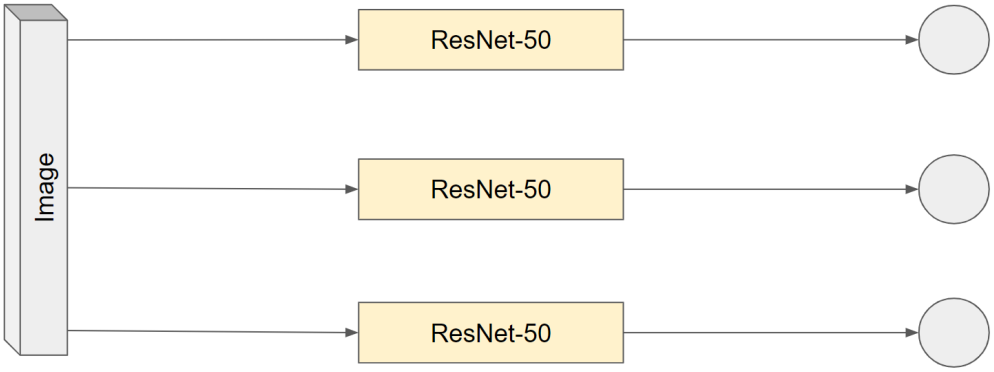
Let $\vec{\varphi}$ denote the features extracted by the ResNet backbone. This vector is then passed to B branches. Each branch predicts a single hierarchical level and consists of a classifier that takes $\vec{\varphi}$ and calculates the logits $\vec{z}_b$ for branch b. The logits are a vector of length C_b, where C_b is the number of classes in the branch b. Let the weight matrix be denoted as $\mathbf{W}_b$ and the bias vector as $\vec{u}_b$. Then, the predicted categorical distribution $\vec{y}_b$ for the b-th branch can be computed as:

$$\vec{y}_b = \text{softmax}(\mathbf{W}_b\vec{\varphi} + u_b) \tag{5.1}$$

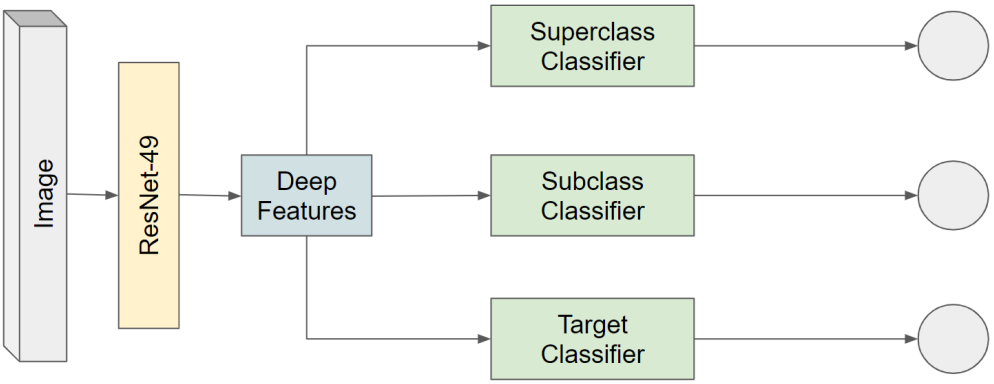
To determine the loss, the predictions are compared to the ground truth labels. Let $\vec{t}_b$ be the ground truth label vector for the b-th branch, where $y_{bj} = 1$ if the j-th class is the correct class and 0 otherwise. The cross-entropy loss $\mathcal{L}_b$ for the b-th branch can be computed as described in 3.4. During training, the weights are learned by minimizing the loss functions. As there are B different losses to backpropagate, they need to be combined into a single loss. This can be done via summation as follows:

$$\mathcal{L} = \sum_{b=1}^{B} \mathcal{L}_b \cdot \lambda_b \tag{5.2}$$

where B is the total number of branches, i.e., the number of hierarchy levels to be predicted, and the weights λ represent how much each loss is weighted. These branch weights will be discussed later. Typically, such weights are chosen to be smaller than one so that the magnitude of the gradient does not get too large.



(a) Architecture of Baseline



(b) Architecture of BSN

Figure 5.1: BASELINE VS BSN ARCHITECTURE. *The baseline model consists of three separate ResNet models, while the BSN shares one. Both models have three separate classifiers.*

This approach provides a general approach to predicting a given hierarchy with an arbitrary number of levels, as a single branch can represent each level. This approach highlights the reasoning for a simplified hierarchical tree with all end nodes belonging to the same level. Otherwise, the calculation of the loss would be much more complex. Using a simplified tree keeps the loss relatively simple and allows for a focus on the architecture of the models. As a consequence of this branch approach, each level of the hierarchy is predicted - somewhat - independently, e.g., a child class may be predicted that belongs to a different subclass than the predicted one.

5.1.3 Information Sharing Across Branches

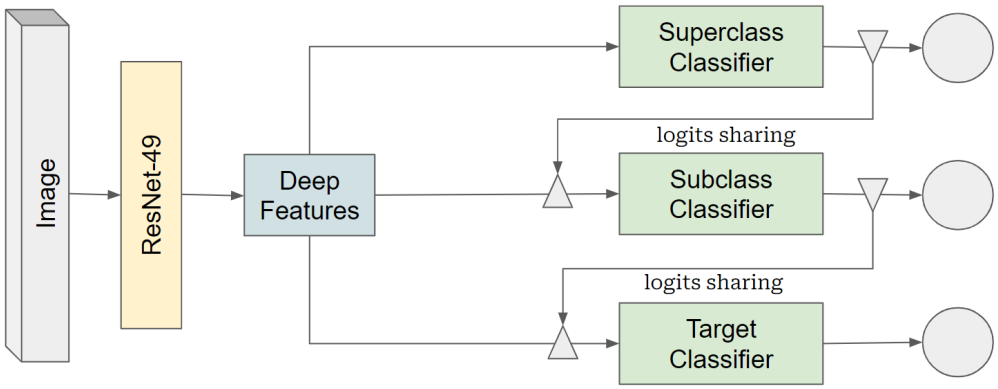
The BSN architecture described in Section 5.1.2 aims to determine how the performance and learning semantics are impacted through joint training. This architecture is relatively simple, and enhancing it may lead to greater performance. In particular, the BSN model does not explicitly share information across branches. The model aims to learn the hierarchy by backpropagating multiple objectives. Consequently, the model jointly optimizes the result in the entire hierarchy. However, it may be desirable to share information on the hierarchy between branches explicitly. As such, modifying the architecture so that branches share their knowledge may be beneficial to boost performance. For example, if the superclass has been correctly identified, the subclass may be much easier to predict. In this thesis, two concrete approaches are taken to enable knowledge sharing between classification branches. The first is the Logit Share-Net (LSN), and the second the Attention Share-Net (ASN). In both network architectures, information from a higher-hierarchical level is shared with the next lower one.

The LSN enables information sharing by sharing the logits of the higher hierarchical level. First, the higher hierarchical level is predicted. The logits of this prediction are then concatenated to $\vec{\varphi}$ when predicting the next layer. The key idea behind this approach is to offer information on the previously predicted hierarchical level to the next. This information could then be used to make predictions that consider the underlying hierarchy for the following hierarchical layer. The base case in (5.1) can now be extended as follows for the b-th branch:

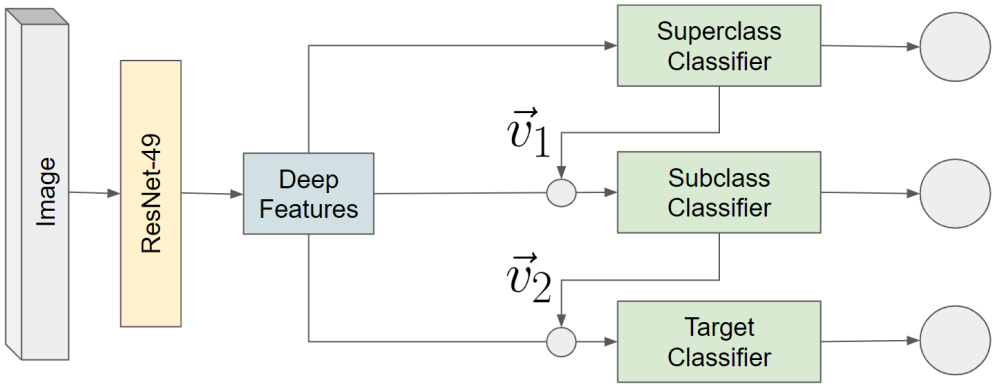
$$\vec{z}_b = \mathbf{W}_b \operatorname{concat}(\vec{\varphi}, \vec{z}_{b-1}) \tag{5.3}$$

Note that the softmax function is still applied to calculate CCE. However, the raw logits are appended as they represent unnormalized outputs, which are appended to the unnormalized deep features $\vec{\varphi}$. Where $\operatorname{concat}(\vec{\varphi}, \vec{z}_{i-1})$ represents a concatenation of the deep features and the logits of the previous branch. This results in an input vector of size $D + C_{b-1}$, where D is the number of input dimensions (i.e., the dimension of the vector $\vec{\varphi}$). Note that the bias has been left out for a simpler notation. The idea is that sharing the previous predictions helps to make better predictions for lower levels. The reason why higher levels are predicted first is that it tends to be easier to predict fewer classes than a large number of classes, which in turn may improve learning. The network may learn how the shared probability relates to the prediction of the next lower level, thus enabling it to learn the relation between two hierarchical levels. The logits are used instead of a one-hot encoded vector to allow for consideration of higher-level predictions without limiting the available information to the class with the highest probability. The obvious downside is that appending probability distributions leads to a larger number of parameters, as the input size of the classifiers is larger and thus requires a larger weight matrix.

Attention mechanisms in CCNs inspired the ASN, more precisely, the idea of transforming the deep features so that relevant features are highlighted to improve performance. Each hierarchical level has a weight matrix $\mathbf{W}_b$ that classifies $\vec{\varphi}$ into a categorical distribution. The key idea is to use the weight matrix of a higher hierarchical level to transform $\vec{\varphi}$ for the next lower hierarchical level. The weight matrix $\mathbf{W}_b$ contains C_b rows resulting in the output $\vec{z}_b$ for a given hierarchical level b. Using the softmax function, the logit vector $\vec{z}_b$ can be normalized to obtain $\vec{a}_b$. $\mathbf{W}_b$ should



(a) Architecture of LSN



(b) Architecture of ASN

Figure 5.2: LSN AND ASN ARCHITECTURES. *The LSN architecture takes the logits of the previous hierarchical level and concatenates them. The ASN architecture takes the weights matrix of the previous classifier, sum the weights up using the probabilities, and then fuses the features through multiplication.*

be used to transform the features $\vec{\varphi}$ so that new features are obtained to the next lower hierarchical classifier. Particularly, the weight matrix can be adjusted with the normalized probability vector $\vec{a}_b$:

$$\mathbf{P}_b = \vec{a}_b \cdot \mathbf{W}_b \tag{5.4}$$

In this new matrix $\mathbf{P}_b$, each row is now weighted with the probability. If a given class has a low probability, the weights used to predict that class $\mathbf{W}_b$ are weighted with that low probability. Each element of the same column of this matrix is then summed up to obtain a vector $\vec{v}_b$. This vector transforms the deep features $\vec{\varphi}$. The column dimension is D, i.e., the dimension of the vector $\vec{\varphi}$. In mathematical terms, a vector $J_{1,C} = (1, ..., 1)$ can be used to obtain such a summation.

$$\vec{v}_b = J_{1,C} \cdot \mathbf{P}_b \tag{5.5}$$

This results in a $1 \times D$ column vector. Each element of $\vec{v}_b$ is the sum of elements in the corresponding column of $\mathbf{P}_b$. This vector can then be used to transform the features for child classes of s in the next hierarchical level $b + 1$.

$$\vec{\varphi}'_{b+1} = \begin{cases} \vec{\varphi} \odot \vec{v}_b & \text{if b > 1} \\ \vec{\varphi} & \text{if b = 1} \end{cases} \tag{5.6}$$

Multiplying the deep features $\vec{\varphi}$ with the attention weight vector v_b from the higher hierarchical level results in the new features $\vec{\varphi}'_{b+1}$. The same $\vec{v}_b$ is applied to the deep features of all samples belonging to that parent class. These transformed features include information on which features are relevant for classification in the previous (i.e., higher) branch. Consequently, now each hierarchical layer gets different deep features to be classified - one that is based on the weights of the higher-level classifier and the predicted probability distribution of a given sample. As in the previous approaches, these features are then classified by an FC layer. Rather an appending information on which higher class the sample likely belongs to, like in the LSN, this approach highlights relevant parts of the features by integrating information from the higher-level classifier. The proposed mechanism aims to allow the network to learn the underlying hierarchy by transforming the features in such a manner that features which were deemed important by the higher-level classification layer are highlighted in the features of the lower hierarchical level. The transformed features $\vec{\varphi}'_{b+1}$ differs from sample to sample based on the probability distribution of the higher-hierarchical level since both v_b and $\vec{\varphi}$ vary between each sample. Still, samples with similar higher-level predictions, and thus similar v_b, similarly transform the features, hopefully allowing the network to differentiate between samples of different hierarchical classes. Note that despite no feature transformation happening at the highest hierarchical level, it may still learn from this process. This is because during the computation of the transformation vector v_b, the computation graph remains enabled. Thus the backpropagated gradient is influenced by the transformation that happens.

5.2 Addressing Class Imbalance

The ImageNet data set is fairly balanced regarding the sample distribution for each class. This means that the target classes have only a limited issue with class imbalance, especially since the validation set size is equal for each class. However, as soon as a simplified hierarchy is constructed as described in Section 4.3, this may not hold for all hierarchical levels. The superclass *artifact* contains 56.8% of all target classes, while the superclass *living things* only contains 43.2% Class imbalance is an even larger issue for the subclasses. The largest subclass *device* contains more than 18% of all target classes, while the smallest class *implement* only contains 5%. While

this may be dealt with by selecting a different hierarchical split (e.g., by breaking up the *device* class further) or dropping classes from large classes, there are other options to deal with class imbalance. Such an imbalance can lead to problems with classification, as the number of samples for superclasses or subclasses may vary greatly. The issue arises because a given classifier classifies instances of less represented classes less accurately (Johnson and Khoshgoftaar, 2019). A majority class may dominate the learning process due to its increased prior probability, while a minority class is learned to a lesser degree. However, this thesis aims to take an approach where even imbalanced hierarchies can be learned, as it is unlikely that any real-world scenario has a naturally balanced distribution. This work considers class weights in the loss function to limit the effects of class imbalance. These weights correspond to the inverse prevalence of each class, reflecting the proportion of samples belonging to a given class. A weight $w_{c,b}$ is calculated for each class c on the same hierarchical level b:

$$w_{c,b} = \frac{N_b}{C_b N_{c,b}} \tag{5.7}$$

N_b represents the number of total training samples in C_b classes. $N_{c,b}$ is the number of training samples in a given class c in the hierarchical level b. This process can be done independently of the hierarchical structure. This means the number of *subclasses* is not considered. Consequently, the weights are determined solely by the distribution of samples across classes. The CCE introduced in (3.4) can be modified to incorporate these weights to calculate the loss for each hierarchical level b:

$$\mathcal{L}_b^{CCE} = -\frac{1}{N_b} \sum_{n=1}^{N_b} \sum_{c=1}^{C_b} w_{c,b} t_{n,c} \odot \log y_{n,c} \tag{5.8}$$

This is implemented through the `scikit-learn` function.[2] Weighted CCE compels the model to pay more attention to minority classes. This should be reflected in the choice of evaluation metric. While *accuracy* is a popular evaluation metric, *balanced accuracy* offers a measure of performance that is not favoring good classification of a majority class (Johnson and Khoshgoftaar, 2019). True positives (TP_c) describe samples correctly classified to belong to class c, while false negatives (FN_c) refer to samples that have been misclassified as a different class than c.

Accuracy is defined as follows:

$$\text{Accuracy} = \frac{\sum_c^C TP_c}{\sum_c^C (TP_c + FN_c)} \tag{5.9}$$

This can lead to issues, as a majority class may greatly dominate this measurement. For example, imagine a scenario where out of three classes, one contains 9'000 samples while the other two have 500 each. If one only measures the accuracy of a classifier, it could blindly classify all samples as the first majority class. In this example, the classifier reaches an accuracy of 90%, even though all samples of the other two classes were misclassified. The related metric of balanced accuracy is defined as follows

$$\text{Recall} = TPR = \frac{TP}{TP + FN} \tag{5.10}$$

$$\text{BA} = \frac{1}{C} \sum_c^C \frac{TP_c}{TP_c + FN_c} \tag{5.11}$$

In these calculations, while we do consider instances from all classes, TP_c and FN_c specifically pertain to instances associated with the class c. Averaging the recalls of each class provides a

[2] https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html

measure that is less biased toward majority classes. In the above-described scenario, the balanced accuracy is only 33%. One class is perfectly classified, while the others are completely incorrectly classified. By accounting for how well each class is classified, a more nuanced metric is gained. Note that for balanced classes, i.e., classes with a similar number of samples, balanced accuracy tends to converge with regular accuracy. This means that for the target classes in the validation set, the balanced accuracy and accuracy are the same, as all classes have the same number of samples.

5.3 Evaluation Approaches

A key question for any classification task is properly evaluating the approach taken. In Section 5.2, it was already discussed how class imbalance might impact accuracy as an evaluation metric. As such, the models are trained using weighted cross entropy in (5.8) and evaluated using balanced accuracy as seen in (5.11). However, since the hierarchy consists of multiple levels with different classifiers, it makes little sense to calculate the balanced accuracy over classes of different hierarchical levels, as each sample belongs to one class at each level. Instead, there will be one balanced accuracy score per hierarchical level. So each hierarchical level has its own balanced accuracy:

$$\mathrm{BA}_b = \frac{1}{C_b} \sum_{c=1}^{C_b} \frac{TP_c}{(TP_c + FN_c)} \tag{5.12}$$

This is a fair evaluation metric for a given hierarchical level, as only the classes C_b of a hierarchical level are considered. As such, these levels are looked at separately, resulting in three balanced accuracy metrics, one per hierarchical level. This same principle is applied to the hierarchical weights described in (5.7). The class weights are applied to the CCE function of each hierarchical layer.

5.3.1 Confusion Matrices

Confusion matrices are a popular tool for identifying frequent confusion between classes and have been used to gain insights into classifiers for ImageNet (Bilal et al., 2017). In a multiclass scenario, one can measure how many samples were predicted as the actual class and how many as a different (incorrect) class. Table 5.1 shows an example of this. One of the axes encodes the predicted class, the other on the actual (ground truth) class. The diagonal of this matrix thus represents the number of correctly classified samples. Note that this depiction also visualizes how the per-class recall required for (5.11) is calculated: In each row, the right prediction (TP) is divided by the total number of samples in that row (TP + FN). More interestingly, such a confusion matrix provides insights into how well a given hierarchy is learned. This is because, similarly to Bilal et al. (2017), one can reorder the axes (i.e., the classes) so that the order of the classes corresponds to the hierarchy. In practice, this means that subclasses of the same superclass should be ordered next to each other.

For instance, in a two-level hierarchy, subclasses 1 and 2 are under superclass A, while 3 and 4 are under B. Table 5.1(a) and Table 5.1(b) show confusion matrices for these classes and superclasses. The subclass classifier might mispredict within the same superclass, indicating it recognizes superclass boundaries. This suggests the model understands the hierarchy, distinguishing subclasses within the same superclass more than those from different ones.

When classifying the ImageNet set, the resulting confusion matrix may be hard to interpret due to the large number of classes to be classified. In particular, there are 660 classes on the lowest level. This makes a detailed comparison of confusion matrices challenging, as each class is

	Predicted Class			
	1	2	3	4
Actual Subclass 1	85	10	0	0
Actual Subclass 2	15	80	0	5
Actual Subclass 3	5	0	70	30
Actual Subclass 4	0	10	5	85

(a) Example Confusion Matrix depicting classification according to subclasses 1-4

	Predicted Class	
	A	B
Actual Superclass A	185	10
Actual Superclass B	25	180

(b) Example Confusion Matrix of superclasses A and B

Table 5.1: EXAMPLE CONFUSION MATRICES. *Here, an example of a confusion matrix is provided. They represent exemplary results from two classifiers, classifying two different hierarchical levels.*

only represented by a tiny square. However, it still makes it possible to recognize block structure which (ideally) corresponds to the underlying hierarchy. Furthermore, the most interesting part of analyzing how well the hierarchy is learned is looking at which higher-level hierarchical class misclassifications belong. This means that one can look at the total misclassifications of classes that are a part of the same super- or subclass. In the example presented in Table 5.1, one can see that 25 out of 30 mistakes happen inside the right superclass A, and 35 out of 50 inside the right superclass B. In total, 60 out of 80 mistakes happen inside the correct superclass. The confusion matrices can also generalize from a two-level to a three-level hierarchy by considering the distribution of the lowest level in relationship to both hierarchical classes: Additionally to the superclasses, subclasses can be used. The target classes (i.e., the lowest level) can then be evaluated for mistakes.

5.3.2 Semantic Conditioning

With these evaluation metrics, one can measure how well the ground truth hierarchy can be classified. However, this gives little information on how the resulting network utilizes the provided hierarchy. One possibility to gain insight into this issue is to measure when a child class of a different parent class is predicted. In multiclass classification, a neural network predicts the probability that a sample belongs to any of the existing classes. Based on this categorical distribution, the most likely class is chosen as the final prediction. However, there are valuable insights by looking at how the predicted probabilities relate to the hierarchy to obtain information on the learning semantics of a given network. The key idea is that these probabilities can be sorted to determine where the first hierarchical mistake happens. In this setting, a hierarchical mistake is when a child class is predicted that belongs to a different parent class. Assume the hierarchy described in Table 5.1. Imagine if the predicted distribution for a given sample of class 1 is $[0.4, 0.1, 0.3, 0.2]$, i.e., a network predicts the sample to belong to class 1 with a probability of 40%, class 2 with 10%, and so on. While the sample is correctly classified as class 1, it still struggles to learn the decision boundary between the two superclasses. The output probabilities can be $\mathrm{argsorted}$ in descending order. Argsorting is a process that returns the indices of an array that would sort it in descending order. Here the resulting array is $[1, 3, 4, 2]$. Now, to measure this phenomenon, one can identify the first index of an incorrect superclass, i.e., class 3 at position 2.

$$\vec{q} = \mathrm{argsort}(\vec{z}) \tag{5.13}$$

In this scenario, $\vec{z}$ are raw logits and not a normalized probability distribution as described in the example above. While such a normalized distribution can be obtained via the softmax function, the resulting vector $\vec{q}$ is the same for both approaches, as the relative ordering does not change through the softmax function.

Formally, let P_t denote the target parent class and C_t denote its target child class. There are S child classes of P_t (including C_t itself), denoted by C_s where $s = 1, \ldots, S$. U denotes the number of unrelated child classes at the same hierarchical level, which belong to different parent classes. These children are denoted as C_u where $u = S + 1, \ldots, U$. The neural network generates logits $\vec{z}$ for $S + U$ classes at a given hierarchical level, representing the probabilities for the child classes.

If the neural network learns from the provided hierarchy, it should be able to learn the decision boundaries between the child classes of different high-level classes. Effectively, this means it learns that the lower-level classes S belong to the same parent class and not to an unrelated one. One can measure this by finding the first instance in the argsorted vector $\vec{q}$ that corresponds to a class not associated with the ground truth parent class P_t. The higher the position, the better the network has learned the hierarchy, i.e., the closer the children of the same parent class are to each other inside the classification space. The best performance in this measurement is to predict all S classes before any of the U classes are predicted. To find the first position given the logit vector $\vec{z}$, first, the argsorted vector $\vec{q}$ is needed, where a_i refers to the index of the i-th largest element in $\vec{z}$:

This allows for the definition of the function g that returns the position of the first prediction in the argsorted vector $\vec{q}$, which is not a child of the ground truth parent class P_t.

$$g(\vec{q}, P_t) = \arg\min_i \operatorname{parent}(a_i) \neq P_t \tag{5.14}$$

Here, $\operatorname{parent}(c)$ returns the parent class of a given child class c. The function g finds the minimum index i for which the parent class of the i-th element in the argsorted vector is not equal to the ground truth parent class P_t. Now, the best semantic distinction of a neural network is achieved when $g(\vec{q}, P_t) = S$, i.e., all child classes of P_t are assigned a higher probability than non-child classes. This means that the range of g for a given sample ranges between 0 and S. To interpret and visualize the results g is applied to each sample with its ancestor class.

This look beyond the most probable classification allows some inference on how well a given model learns the decision boundaries between hierarchical classes. One can turn this into a histogram for further interpretation. The histogram of a given class P_t can be calculated by applying $g(\vec{q}, P_t)$ to all $\vec{q}_n$ of samples that belong to P_t. The histogram represents the frequency distribution of the hierarchical mistakes indexed by the function $g(\vec{q}, P_t)$, where each bin refers to the number of first mistakes made at a given index. This can also be depicted as a cumulative distribution, representing how many samples contain a mistake up to a given index. By normalizing all cumulative values by N_{P_t}, i.e., the total number of samples belonging to the parent class, one receives a CDF function. The CDF function $\mathrm{CDF}(i)$ is defined for a single parent class, and represents an empirical approximation on the probability that the predicted categorical distribution of a sample from P_t contains an unrelated child class by index i.

For easier interpretability, a complementary cumulative distribution function (CCDF) is chosen. This is the inverse of the CDF (i.e., $1 - \mathrm{CDF}(i)$) and depicts what percentage of classes do *not* contain a hierarchical mistake at a given index.

The higher the value at any given point in the CCDF, the better. If a CCDF curve stays flat in the beginning and only drops at a later point, then a model makes mistakes further in the back, indicating that the classifier has learned the decision boundaries. In contrast, if the curve drops rapidly, then it frequently contains a prediction of an unrelated child class earlier in its predictions. A network can be seen as semantically conditioned when it learns the semantic relationship of a higher hierarchical class for subordinate classes, i.e., if the curve is flat in the beginning and only drops at the end.

5.4 Open-Set Classification

The described approaches focus on analyzing how well the proposed architectures can discriminate between classes from the provided hierarchy. However, many more classes are not represented in the ImageNet-1k data set. This is illustrated by the fact that the ImageNet-1k only spans 1'000 classes, while the WordNet hierarchy contains 117'000 synsets (Princeton University, 2010). In Open-set classification, unknown samples should be recognized as such. Consequently, it is worth exploring how the proposed models behave when confronted with unknown classes.

This is especially relevant for this thesis as the hierarchy used for classification only contains 660 of the 1'000 ImageNet classes. The rest of these classes are unknown classes to a certain degree. However, not all of the remaining 340 classes are truly unknown, as many of these classes belong to ancestors that are represented in the hierarchy. Only 72 ImageNet classes are neither a part of the *artifact* or *living thing* superclasses. The remaining 268 classes are from known superclasses but from unknown subclasses. Consequently, these two sets of unknown classes should be evaluated separately to examine the impact of semantic distance. Still, issues remain, especially with the classes from the known superclasses, i.e., the unknown subclasses. During the hierarchy construction, there were closely related classes that were not considered in the subclass hierarchy, such as 60 dog breeds that do not belong to the subclass *hunting dog*.

To evaluate how the models perform on open-sets, Open-Set Classification Rate (OSCR) curves are used as proposed by Dhamija et al. (2018). This evaluation is also employed in the work of Palechor et al. (2023), who provide the source code for evaluating their models. [3] This metric is only applied to the lowest hierarchical level for this thesis as an in-depth analysis of the other hierarchical levels is out of the scope of this thesis and less relevant to the research question. As such, the branch index b is left out of the following definitions for better legibility.

Dhamija et al. (2018) define the calculation of OSCR curves as follows. First, the data set is split into known classes $\mathcal{D}_c$ and unknown classes $\mathcal{D}_u$. Let θ be the score threshold. If the maximum predicted probability by the softmax output of a network is smaller than θ, it is rejected as an unknown sample. The False Positive Rate (FPR) is the fraction of samples of $\mathcal{D}_u$ that is classified as any known class $c \in \mathcal{D}_c$ with a probability greater than θ. The Correct Classification Rate (CCR) is calculated as the fraction of samples where the predicted class c corresponds to the ground truth class t_n with a probability greater than θ:

$$\begin{aligned} \mathrm{FPR}(\theta) &= \frac{|\{x_n \mid x_n \in \mathcal{D}_u \wedge \max_c y_{n,c} \geq \theta\}|}{|\mathcal{D}_u|}, \\ \mathrm{CCR}(\theta) &= \frac{|\{x_n \mid x_n \in \mathcal{D}_c \wedge \arg\max y_{n,c} = t_n \wedge y_{n,c} \geq \theta\}|}{|\mathcal{D}_c|}. \end{aligned} \tag{5.15}$$

Here $\max_c y_{n,c}$ corresponds to the highest confidence score predicted for a sample x_n and $\mathcal{D}_c \wedge \arg\max y_{n,c} = t_n$ denotes the correct prediction of the ground truth label t_n. The OSCR curve is plotted by varying the threshold θ between 0 and 1. The implementation for generating the OSCR curves is taken from a slightly adapted version of Palechor et al. (2023).[4] In this version, the FPR and CCR are calculated using a $\geq$ instead of a $>$, which means that this implementation also considers the FPR and CCR for confidence values equal to 1. The thresholds are selected based on all the occurring values of $\max_c y_{n,c}$. Just as in the work of Palechor et al. (2023), these curves are plotted with logarithmic FPR axes and linear CCR axes.

[3] https://github.com/AIML-IfI/openset-imagenet
[4] https://github.com/AIML-IfI/openset-imagenet-comparison

Chapter 6

Experiments and Results

Several experiments are conducted using the approaches described in Chapter 5. Different architectures are explored and compared with each other to see how the architectures perform.

6.1 Set-up for experiments

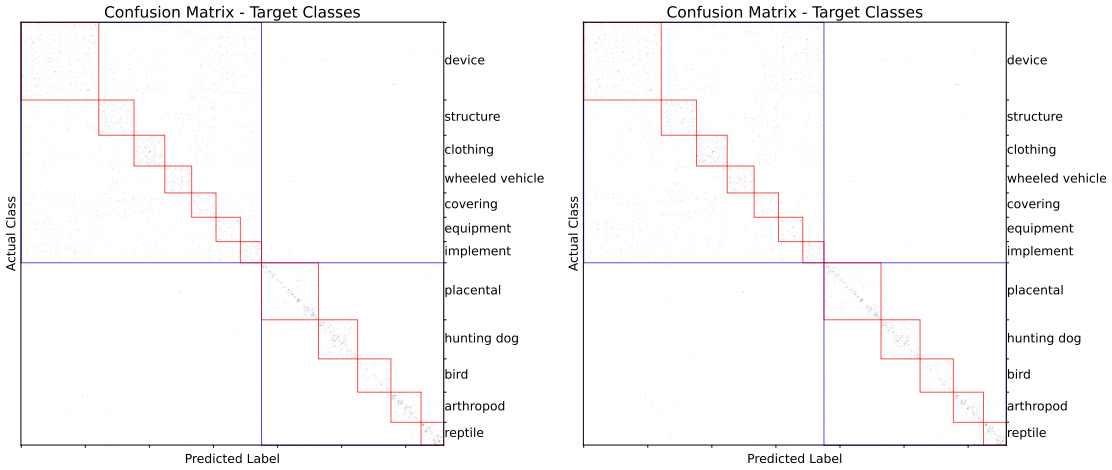
For a fair comparison between models, all models are trained with the same hyperparameters. The networks are implemented in `pytorch`.[1] All implementations use the `torch.optim.SGD` optimizer with the learning rate set to 0.005, the momentum to 0.9, and the weight decay to 0.0005. This scheduler decays the learning rate by 0.1 every 30 epochs. The networks are trained for 66 epochs. Furthermore, StepLR was used as a learning rate scheduler. This further helps to reduce the learning rate to find the minimum. Due to the large number of images, batching is needed as the images do not fit into the GPU memory. The batch size was 80 for training and 50 for validation. For faster training, the training was conducted on a NVIDIA GeForce RTX 2080 Ti GPU. For image preprocessing, the shorter side of the image is resized to 256 pixels, and then a crop of size 224×224 is taken. This crop is random for training images to increase variation and make the module more robust. For the validation images, the center crop is taken so that the evaluation process is reproducible. A random horizontal flip is also applied to the training images to make the model more resistant to transformations. The color channels of all images are also normalized with the values mean $= [0.485, 0.456, 0.406]$ and std $= [0.229, 0.224, 0.225]$ taken from the `pytorch` ResNet-50 implemenation.[2] These values have been calculated from the ImageNet data set. They are commonly used to normalize the ImageNet data.

6.2 Shared Backbone

As a first experiment, a comparison between a single shared network and three separate baseline networks is conducted. This experiment aims to answer the research question of how the performance of three single-objective networks compares to that of a single multi-objective network. To measure this, the balanced accuracy scores of each hierarchical layer are compared. The results on the validation set of all three hierarchical levels are depicted in Table 6.1. The BSN outperforms the baseline on all hierarchical levels, showing how the model benefits from a shared backbone. The increased performance is likely because the shared backbone of the BSN extracts features relevant to all three hierarchical levels, levering the correlation between hierarchical levels.

[1] https://pytorch.org/

[2] https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet50.html



(a) Confusion Matrix Baseline

(b) Confusion Matrix BSN

Figure 6.1: CONFUSION MATRICES. *Only misclassifications of the target classes are represented in this confusion matrix. The 12 red borders represent the subclasses, and the two blue borders are the superclasses. Mistakes are colored in grey; the darker the shade, the more frequently the same mistake occurs. Note that the color is in a log scale.*

Table 6.1: VALIDATION BALANCED ACCURACY BSN AND BASELINE. *This table reports the balanced accuracy of the validation set after 66 epochs of training. The values in bold indicate the highest balanced accuracy score.*

Model	BSN	Baseline
Superclass	**98.90**%	97.77%
Subclass	**90.80**%	87.09%
Target Class	**76.19**%	75.51%

To gain further insights, one can look at the confusion matrix to see whether the mistakes in the target classes happen inside or outside the higher hierarchical classes. Because correct classifications tend to dominate the color scheme, only misclassifications are depicted in Figure 6.1(a) and Figure 6.1(b). In these figures, the two superclasses are marked by a blue border and subclasses by a red border. Furthermore, a log scale was used to color the confusion matrices, as otherwise, a few frequent mistakes dominate the color scheme. Ideally, mistakes should happen inside the correct super- and subclasses. Visually, these confusion matrices look nearly identical. However, there are clear differences inside the superclasses. While the target classes are classified correctly as the *artifact* superclass – indicated by the blue border – many mistakes happen outside the correct subclass. Compare this to the *living things* superclass. Mistakes tend to happen both inside the right super- and subclass. The mistakes of both models are close to the diagonal line, meaning that mistakes happen in closely related classes. This effect can be seen in both Figure 6.1(a) and Figure 6.1(b), the *artifact* subclasses are in the top left, the *living things* in the bottom right. This can be somewhat quantified as described in Section 5.3 by measuring how

many misclassifications of target classes happen inside the correct super- or subclass, i.e., are misclassified as a target class of the same correct super- or subclass. This is given as a percentage of misclassification to be consistent with the confusion matrix, where only misclassifications of target classes are depicted. For the baseline model, 99.83% of all misclassifications of target classes occur in the same superclass. For the BSN, it is slightly lower at 99.82%. This means that only a small percentage of target classes that belong to one superclass are classified as target classes of a different superclass. The BSN also classifies 98.82% of all misclassified target classes as a target class of the same subclass compared to the baseline, which performs worse with 98.60%. This means that even without any knowledge of the hierarchy, the vast majority of mistakes happen in semantically related classes. However, this metric is somewhat flawed as it does not consider the absolute number of mistakes. As such, this metric is only provided for these two models to highlight that the baseline can learn the semantic hierarchy based on visual similarity, at least to a similar degree as the BSN. From these confusion matrixes, it can also be concluded that this visual similarity is stronger for subclasses of the *living things* superclass. Still, there certainly is visual similarity inside the *artifact* superclass, as many mistakes happen inside this superclass, which is already desirable. However, the *artifact* subclasses are not as visually distinguishable from each other as the *living things* subclasses.

More information can be obtained by looking at the probabilities generated for each sample. This measures how semantically conditioned the resulting networks are. As seen in Figure 6.2, the BSN predicts a target class belonging to a different parent later than the baseline. These curves represent a CCDF to allow for easier interpretation as the sizes of samples in these subclasses vary just as the number of target classes inside the subclass. By normalizing the x-axis with the number of target classes inside a subclass, one gets values between 0 and 1 for each subclass. This means if at 0.2 of the x-axis, the y-axis reads 0.8, then 80% of samples do not contain a mistake when looking at the index that is 20% of the subclasses. Assume such a subclass has 40 target classes, each containing 50 samples. Then, 1'600 of these samples do not contain a mistake by the 8th index. While not straightforward to interpret, normalizing both axes allows for comparisons across subclasses. Note that the curves are not smooth as they represent a discontinuous space, i.e., the index of the first misclassification. Both curves also nearly start at 1 since only a small fraction of all misclassified target classes are classified as a target class from a different sub- or superclass. Misclassifications generally only represent a fraction of all samples. Thus, only very few mistakes happen at the first index.

The first 7 subplots in these two figures belong to the superclass *artifact* and the latter five to the superclass *living things*. Similarly to the misclassifications observed in the confusion matrix, a difference between the subclasses of the two superclasses emerges. Subclasses of *artifact* show a steeper drop at lower indices, while the curves of subclasses from the *living things* superclass are flatter initially. This pattern can be seen in both models. Both lines are nearly 1 From these results, it is clear that the BSN outperforms the baseline in all three tasks regarding balanced accuracy. The results also indicate that the BSN's semantic conditioning is slightly better than the baseline's.

6.3 Information Sharing Models

From the results in Section 6.2, it is clear that the model greatly benefits from a shared backbone. Based on the relatively simple architecture of the BSN, more advanced architectures can be built. As described in Section 5.1.3, the BSN model is expanded with mechanisms to share information between branches and thus (hopefully) improve performance. Namely, two models are trained, one using logits to convey information about the higher hierarchical level, LSN, and the other using an attention mechanism, ASN. This experiment explores whether these two modifications

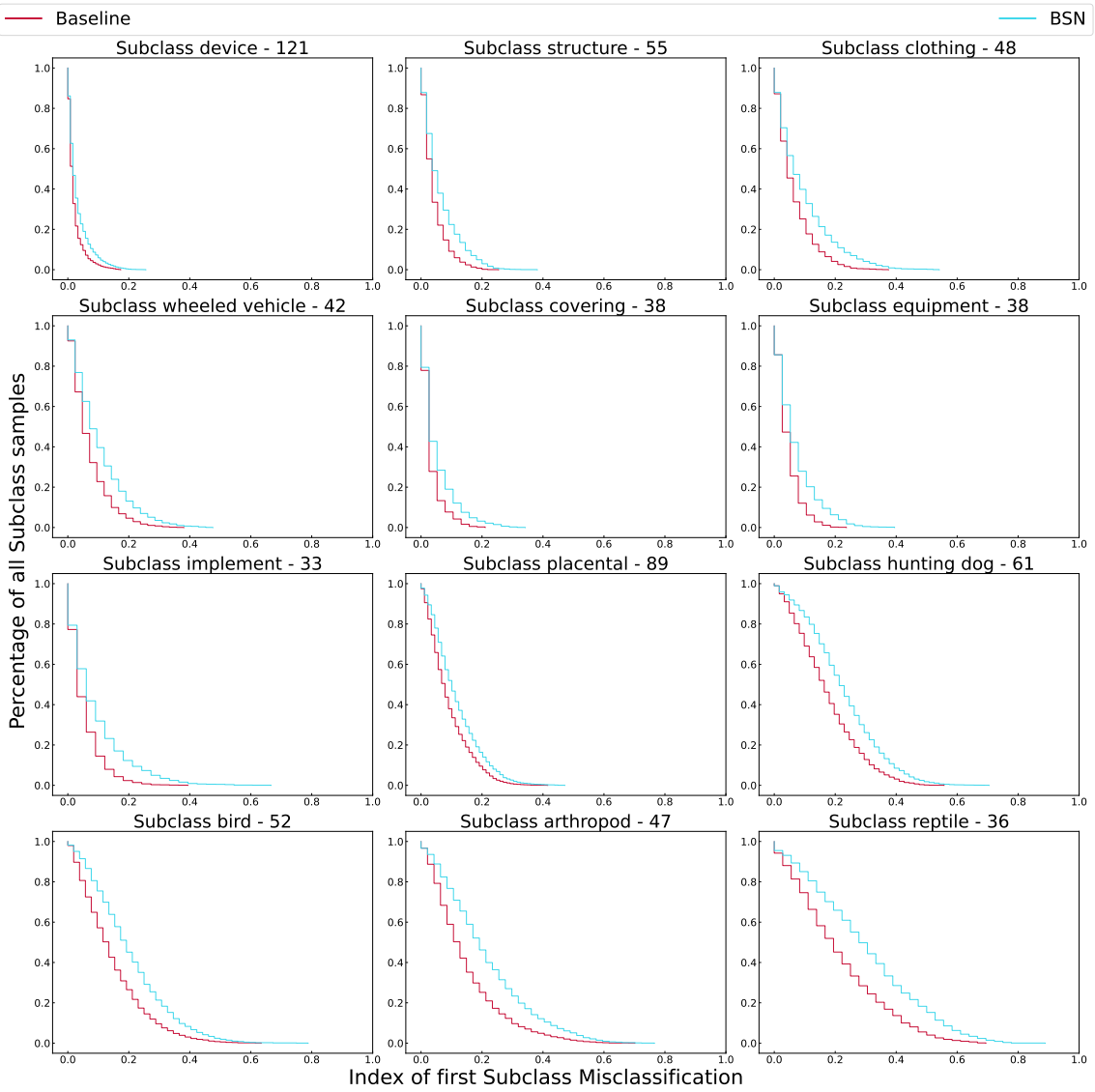


Figure 6.2: SEMANTIC CONDITIONING BASELINE AND BSN. *For the BSN and baseline model, it is measured at which index a severe semantic mistake was made, i.e., at which position a class of a different subclass is predicted, as described in Subsection 5.3.2.*

Table 6.2: VALIDATION BALANCED ACCURACY BSN, LSN, AND ASN. *This table reports the balanced accuracy of the validation set after 66 epochs of training. The values in bold indicate the highest balanced accuracy scores.*

Model	BSN	LSN	ASN
Superclass	**98.90**%	98.85%	98.75%
Subclass	**90.80**%	90.32%	89.25%
Target Class	**76.19**%	75.45%	75.93%

for information sharing can increase performance compared to the BSN.

It is important to note that the LSN has more parameters, as in each branch now $D + C_{b-1}$ input parameters, where C_{b-1} come from the predictions of the higher hierarchical level, and D is the dimension of the deep features $\vec{\varphi}$. This leads to an increase of $O_{b-1} \times O_b$ parameters per hierarchical level, with the first level remaining unchanged as there is no higher hierarchical level to consider. This means the LSN model has 24'897'002 parameters compared to the 24'889'058 parameters of the ASN and BSN.

The balanced accuracy scores are listed in Table 6.2. The proposed architectures perform worse than the BSN on each hierarchical level. The biggest difference in performance is for the subclass classification, where the ASN achieves a balanced accuracy score that is 1% lower than that of the BSN. Otherwise, the performance of the ASN and BSN is comparable, while the LSN lacks behind in target class classification.

The confusion matrices do not show any visible differences and are nearly identical to those presented in Figure 6.1(a) and Figure 6.1(b). Consequently, they are only listed in Appendix A.2.

The question of whether the semantic relation between classes can be learned cannot be answered with the previous metrics. To gain further insights, the networks' semantic conditionings are depicted in Figure 6.3. Again, there is a general pattern of a steeper curve for subclasses of the *artifact* superclass. Here the BSN and LSN perform similarly, while the ASN makes mistakes slightly further back in its predictions. Interestingly, the ASN is much better at differentiating between target classes belonging to the superclass *living things* than the BSN and LSN. To quantify this observation, a table of different reference points can be found in Tables 6.3 and 6.4. In these tables, the probability is quantified that a sample of a target class from a different subclass has been predicted. The reference points are again taken as a percentage of the number of subclasses. In Table 6.3, the subclasses of the *artifact* superclass are displayed. Here the difference between the models is comparatively small. However, as seen in Table 6.4, the difference in subclasses of *living things*, the difference between the ASN and other models is much greater. The ASN thus tends to contain an incorrect prediction later than the other two models.

The accuracy scores of both the ASN and LSN show that the proposed architectures are not effective at increasing balanced accuracy. The LSN has more parameters *and* performs worse than the BSN, making it an ill-suited candidate for further experiments. However, the ASN shows promises when looking at its semantic conditioning. This increased semantic conditioning indicates that the attention mechanism works to a certain extent. As such, further modifications can be made to the architecture to improve both balanced accuracy and semantic conditioning.

6.4 Extending the ASN architecture

The ASN is a unique architecture with a mechanism that is worth further exploration. Consequentially, four additional changes to the ASN architecture are made. These modifications aim to answer the question of how the information sharing of the attention mechanism can be effectively

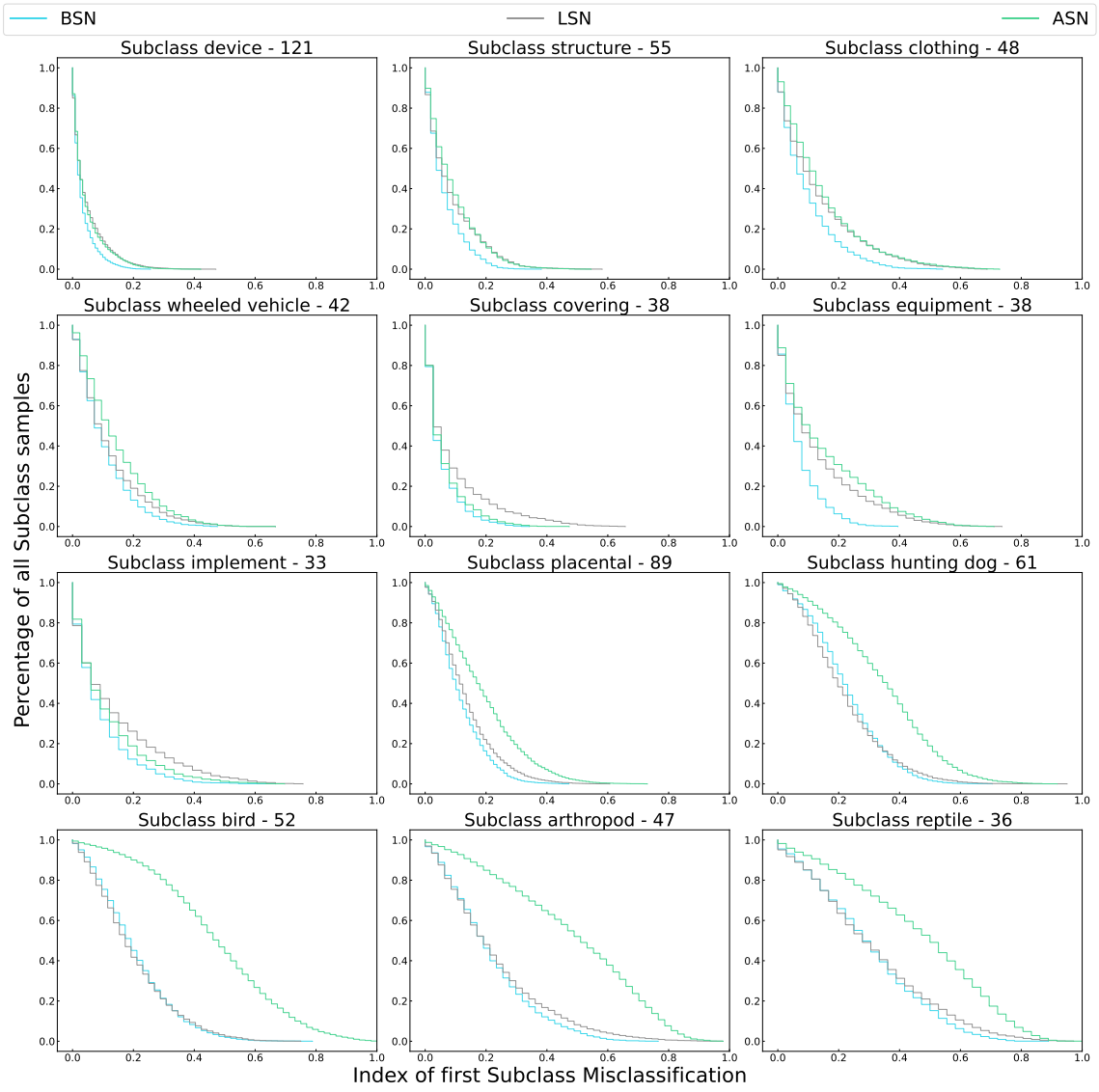


Figure 6.3: SUBCLASS HISTOGRAM OF PREDICTIONS. *A comparison of semantic conditioning of the BSN, ASN, and LSN per subclass.*

Table 6.3: ARTIFACT SUBCLASS CCDF COMPARISON. *The values of the CCDF curve at a selection of different breakpoints, ordered by subclass. The values with the fewest mistakes up to that index are highlighted in bold.*

Index	BSN	LSN	ASN
device			
10%	0.06	0.06	**0.12**
25%	0.00	0.00	**0.01**
50%	**0.00**	**0.00**	**0.00**
75%	**0.00**	**0.00**	**0.00**
structure			
10%	0.22	0.24	**0.37**
25%	0.01	0.01	**0.06**
50%	**0.00**	**0.00**	**0.00**
75%	**0.00**	**0.00**	**0.00**
clothing			
10%	0.40	0.41	**0.55**
25%	0.07	0.06	**0.16**
50%	0.00	0.00	**0.02**
75%	**0.00**	**0.00**	**0.00**
wheeled vehicle			
10%	0.40	0.40	**0.53**
25%	0.07	0.08	**0.17**
50%	**0.00**	**0.00**	**0.00**
75%	**0.00**	**0.00**	**0.00**
covering			
10%	0.19	0.18	**0.22**
25%	0.01	0.01	**0.02**
50%	**0.00**	**0.00**	**0.00**
75%	**0.00**	**0.00**	**0.00**
equipment			
10%	0.28	0.30	**0.51**
25%	0.02	0.01	**0.24**
50%	0.00	0.00	**0.03**
75%	**0.00**	**0.00**	**0.00**
implement			
10%	0.32	0.33	**0.37**
25%	0.07	0.06	**0.12**
50%	0.00	0.00	**0.01**
75%	**0.00**	**0.00**	**0.00**

Table 6.4: LIVING THINGS SUBCLASS CCDF COMPARISON. *The values of the CCDF curve at a selection of different breakpoints, ordered by subclass. The values with the fewest mistakes up to that index are highlighted in bold.*

Index	BSN	LSN	ASN
placental			
10%	0.52	0.54	**0.73**
25%	0.07	0.07	**0.28**
50%	0.00	0.00	**0.02**
75%	**0.00**	**0.00**	**0.00**
hunting dog			
10%	0.83	0.81	**0.91**
25%	0.39	0.35	**0.70**
50%	0.02	0.02	**0.19**
75%	0.00	0.00	**0.01**
bird			
10%	0.75	0.75	**0.96**
25%	0.29	0.29	**0.85**
50%	0.02	0.02	**0.44**
75%	0.00	0.00	**0.09**
arthropod			
10%	0.77	0.75	**0.94**
25%	0.36	0.37	**0.81**
50%	0.05	0.05	**0.52**
75%	0.00	0.00	**0.14**
reptile			
10%	0.85	0.83	**0.92**
25%	0.55	0.52	**0.78**
50%	0.15	0.13	**0.49**
75%	0.00	0.00	**0.10**

be used to achieve better classification accuracy and semantic conditioning.

6.4.1 Architectural Modification

The first of these four changes is called ASN-A. ASN-A uses an argmax function instead of a softmax function to get the relevant weights. Instead of multiplying the categorical distribution with the weights $\mathbf{W}_b$, only the most likely row is selected. Given the weights W_b of the FC layer for level b, one selects the predicted class k.

$$k = \text{argmax}(\vec{a}_b) \tag{6.1}$$

Using this index, one can create a row vector that serves as v_b

$$v_b = \mathbf{W}_{k,*,b} \tag{6.2}$$

$\mathbf{W}_{k,*,b}$ is used as a notation to select the k-th row from the weight matrix $\mathbf{W}_b$. The rest of the attention mechanism follows the same mechanism described in (5.6). This version limits the shared information to the most probable class. Consequently, the transformation vector v_b is the same for all samples with the same predicted class. This may reduce noise and thus share only relevant information. However, it might also lead to the sharing of incorrect information if the higher-level class is incorrectly predicted. In such a scenario, the features given to a classifier are transformed with a vector of an incorrect subclass, which may lead to more confusion.

A second network, ASN-T, extends this idea with specialized training. All previous models function the same way during training and validation - of course, with the key difference that the weights are adjusted during training. However, it is also possible to create an architecture that receives additional information during training. The ASN-T receives the ground-truth labels of the super- and subclasses during training. Instead of taking the argmax of the categorical distribution during training as described in (6.1), the ground truth class t is taken. The ground truth class t corresponds to the row index that is used to get v_b from $\mathbf{W}_b$. Using this index, one can get the row vector that serves as $\vec{v}_b$ from $\mathbf{W}_b$:

$$\vec{v}_b = \mathbf{W}_{t,*,b} \tag{6.3}$$

$\mathbf{W}_{t,*,b}$ is used as an annotation that the entire row t of $\mathbf{W}_b$ is taken. The idea behind this is that this may help to learn the real relationship between higher level classes and lower level ones without the problem of down propagation of errors during training. During validation, no ground truth information should be given to the As the learning only happens with the weights of a single class, the argmax is used for validation rather than the softmax. The issue with ASN-T is similar to ASN-A: Especially during validation, mistakes in higher predictions may propagate down.

As such, an alternative network called ASN-TS is proposed. This network works as the ASN-T during training, using the ground truth vector to find $\vec{v}_b$ as described in (6.3). During validation, however, the predicted categorical distribution is used to get $\vec{v}_b$ as described in (5.5). The reasoning is that using the categorical distribution helps to limit severe mistakes, possibly resulting in better performance on unseen samples, while still providing the ground truth labels during training to allow for learning of the hierarchy.

The final modification experimented with is called ASN-U depicted in Figure 6.4. Like the ASN and ASN-A, no difference between the training and validation phase exists. This network inverts the prediction process. Rather than predicting the higher level first and sharing the information with the lower level, the information from the lower-level prediction is used for the higher-level prediction. The inversion of the attention mechanism is particularly interesting because the lowest hierarchical level may still benefit from the feature transformation. Critically,

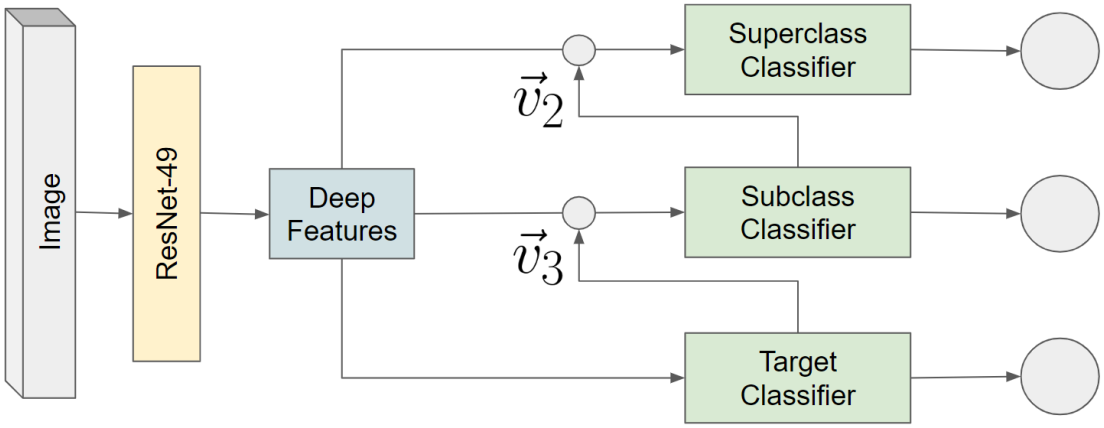


Figure 6.4: ARCHITECTURE OF ASN-U. *The only difference to the ASN is that the sharing mechanism is the other way around.*

during the computation of the transformation vector$\vec{v}_b$, the computation graph remains enabled. This means that the gradients from the higher-level tasks can flow back both into the shared ResNet-49 backbone and the lowest-level classifier during backpropagation, indirectly influencing the learning and performance of the lowest hierarchical level. One reason this is of interest is that despite the reversal of the attention mechanism, the lowest hierarchical level may still learn from the feature transformation.

Turning the attention mechanism upside down may prove effective because if a child class is correctly predicted, then it is certain what the parent class is. Since parent classes have multiple child classes, knowing a parent class may narrow the selection of child classes, but it is never a one-to-one relationship. Furthermore, even if a child class is incorrectly predicted, as long as a sibling class, i.e., a class from the same parent class, is predicted, the mechanism still greatly helps to improve the prediction of higher-level classes. Formally, we can adjust (5.6) so that the weights of the next lower-level classifiers are taken:

$$\vec{\varphi}'_{b-1} = \begin{cases} \vec{\varphi} \odot \vec{v}_b & \text{if b < B} \\ \vec{\varphi} & \text{if b = B} \end{cases} \tag{6.4}$$

The rest of the mechanism works just as in the original ASN mechanism. The goal of this mechanism is to check if reversing the attention mechanism is successful in improving performance and how it affects semantic conditioning.

Overall, these four ASN versions, ASN-A, ASN-T, ASN-TS, and ASN-U, make minor changes to the attention mechanism and compare the results to the first introduced ASN mechanism. The objective of exploring different architectural modifications is to find an effective way to use the proposed attention mechanism. Ideally, an architecture should be found that possesses both good performance and semantic conditioning.

6.4.2 Results of ASN-Variations

The results of the ASN are presented in Section 6.3 do not outperform the BSN architecture in terms of balanced accuracy. Still, the semantic conditioning of the network is increased, indicating that the ASN model is better at learning the semantic relationship of target classes. As such, four additional models, ASN-U, ASN-A, ASN-T, and ASN-TS, are proposed with minor modifications

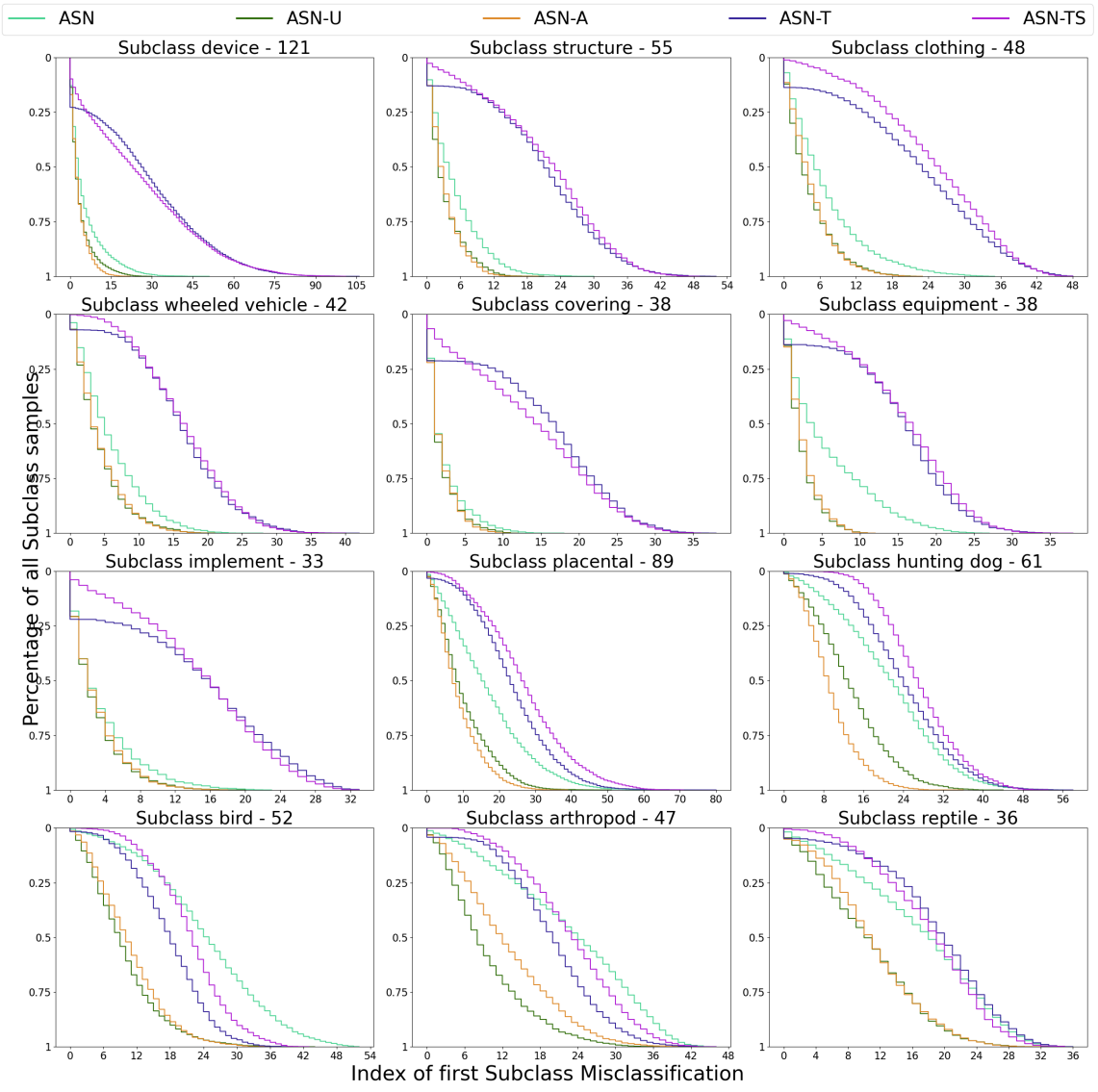


Figure 6.5: SEMANTIC CONDITIONING ASN VARIATIONS. *These CCDF plots compare the semantic conditioning of the ASN, ASN-A, ASN-T, ASN-TS, and ASN-U.*

Table 6.5: VALIDATION BALANCED ACCURACY BSN AND ASN VARIATIONS. *This table reports the balanced accuracy of the validation set after 66 epochs of training. The values in bold indicate the best accuracy score*

Model	BSN	ASN	ASN-U	ASN-A	ASN-T	ASN-TS
Superclass	98.90%	98.75%	**99.92**%	98.87%	98.72%	98.60%
Subclass	90.80%	89.25%	**98.60**%	90.31%	89.32%	89.97%
Target Class	76.19%	75.93%	75.88%	74.29%	72.61%	**76.78**%

to the attention mechanism. The performances are listed in Table 6.5. The ASN-U significantly outperforms the other four ASN versions and the BSN in both super- and subclass classification. The BSN is slightly better than the ASN, ASN-A, ASN-T, and ASN-TS on these two hierarchical levels, but overall their performance is comparable. For the target classes, ASN-TS achieves the best results, outperforming the BSN and the other four ASN versions. Both the ASN-A and ASN-T perform significantly worse in target class classification.

As in previous experiments, semantic conditioning can be analyzed to gain deeper insights into how the model assigns probabilities regarding the hierarchical structure. These histograms are depicted in Figure 6.5. The ASN-T and ASN-TS possess the best semantical conditioning by far. This observation is especially interesting for the *artifact* subclasses. Here, the slope of these two networks is noticeably flatter in the beginning than that of the other three ASN versions. This can also be quantified as seen in Table 6.6 and Table 6.7. For easy comparison, all tested models are listed in this table. Up to the analyzed 75th percentile of the index, the ASN-T, and ASN-TS are consistently less likely to include a prediction of an unrelated *artifact* subclass. The ASN, ASN-TS ASN-T perform much more similarly for subclasses of the *living thing* superclass. The ASN achieves better results in the *bird* and *arthropod* subclasses.

Evaluating which architecture is the best depends on how important the performance on the different hierarchical levels is for a given task. If one is highly interested in determining the correct super- and subclasses, the ASN-U is the best of all analyzed models. The ASN-U turned the attention mechanism upside down; consequently, an improvement when classifying sub- and superclasses is visible, with the target classification remaining similar. The subclass improvement on the validation set is nearly 10% in absolute terms, a significant improvement. It also reaches a near-perfect balanced accuracy score of 99.92% for superclass classification. However, these findings should be enjoyed with caution because there is a significant discrepancy between the training and test accuracy of the ASN-U. This issue will be further discussed in Section 7.1.1. Furthermore, the lowest hierarchical level is most likely the most interesting one - and the most difficult one to classify correctly. Here the ASN-U may compare to other architectures in terms of balanced accuracy but fails as soon as it comes to semantic conditioning.

The other models have more interesting results for the target classes. Both the ASN-T and ASN-A struggle with classification accuracy. This is likely because errors in the higher-level class classification impact the accuracy further down. During validation, the predicted higher hierarchical level may be incorrect. This likely has a strong impact on performance on the lower hierarchical levels, thus making the ASN-T and ASN-A the worst of the proposed networks when measuring balanced accuracy. The implication is that for prediction in the validation set, models profit from integrating uncertainty through the categorical distribution into the transformation vector $\vec{v}_b$ rather than just relying on the most likely class. This can be seen by the fact that the ASN and ASN-TS achieve better accuracy scores for the target classes than the ASN-A and ASN-T.

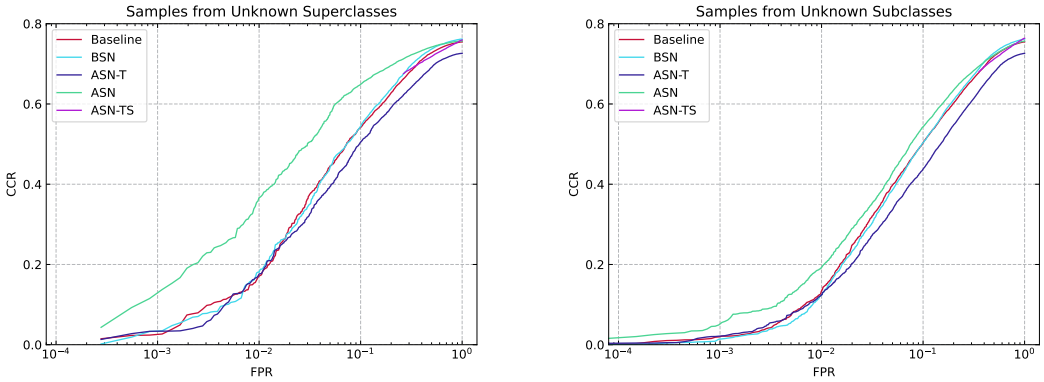
However, what is apparent from the ASN-T and ASN-TS is that providing ground truth labels during training greatly increases semantic conditioning, as the semantic conditioning of the ASN-T and ASN-TS is greatly increased. These networks are trained with ground-truth labels to select

Table 6.6: ARTIFACT SUBCLASS CCDF COMPARISON. *The values of the CCDF curve at a selection of different breakpoints, ordered by subclass. The left-most column depicts which index was used as a percentage of all classes in a given subclass. The other columns represent different models. The values with the fewest mistakes up to that index are highlighted in bold.*

Percentage	Baseline	BSN	LSN	ASN	ASN-U	ASN-A	ASN-T	ASN-TS
device								
10%	0.02	0.06	0.06	0.12	0.05	0.02	**0.70**	0.65
25%	0.00	0.00	0.00	0.01	0.00	0.00	**0.43**	0.39
50%	0.00	0.00	0.00	0.00	0.00	0.00	**0.07**	**0.07**
75%	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
structure								
10%	0.09	0.22	0.24	0.37	0.21	0.20	0.86	**0.90**
25%	0.00	0.01	0.01	0.06	0.01	0.00	0.75	**0.76**
50%	0.00	0.00	0.00	0.00	0.00	0.00	0.27	**0.32**
75%	0.00	0.00	0.00	0.00	0.00	0.00	**0.02**	0.01
clothing								
10%	0.25	0.40	0.41	0.55	0.38	0.42	0.86	**0.96**
25%	0.01	0.07	0.06	0.16	0.05	0.04	0.77	**0.86**
50%	0.00	0.00	0.00	0.02	0.00	0.00	0.45	**0.54**
75%	0.00	0.00	0.00	0.00	0.00	0.00	0.11	**0.14**
wheeled vehicle								
10%	0.23	0.40	0.40	0.53	0.38	0.39	0.93	**0.98**
25%	0.02	0.07	0.08	0.17	0.07	0.07	0.80	**0.81**
50%	0.00	0.00	0.00	0.00	0.00	0.00	0.21	**0.24**
75%	0.00	0.00	0.00	0.00	0.00	0.00	**0.02**	0.01
covering								
10%	0.08	0.19	0.18	0.22	0.18	0.18	0.79	**0.83**
25%	0.00	0.01	0.01	0.02	0.01	0.00	**0.74**	0.66
50%	0.00	0.00	0.00	0.00	0.00	0.00	**0.35**	0.30
75%	0.00	0.00	0.00	0.00	0.00	0.00	**0.05**	0.04
equipment								
10%	0.12	0.28	0.30	0.51	0.23	0.26	0.86	**0.93**
25%	0.00	0.02	0.01	0.24	0.01	0.00	**0.80**	**0.80**
50%	0.00	0.00	0.00	0.03	0.00	0.00	0.27	**0.33**
75%	0.00	0.00	0.00	0.00	0.00	0.00	**0.02**	**0.02**
implement								
10%	0.14	0.32	0.33	0.37	0.33	0.36	0.78	**0.90**
25%	0.01	0.07	0.06	0.12	0.06	0.06	0.72	**0.79**
50%	0.00	0.00	0.00	0.01	0.00	0.00	**0.47**	**0.47**
75%	0.00	0.00	0.00	0.00	0.00	0.00	**0.15**	0.11

Table 6.7: ARTIFACT SUBCLASS CCDF COMPARISON. *The values of the CCDF curve at a selection of different breakpoints, ordered by subclass. The left-most column represents the breakpoint, and the other columns represent different models. The values with the fewest mistakes up to that index are highlighted in bold.*

Percentage	Baseline	BSN	LSN	ASN	ASN-U	ASN-A	ASN-T	ASN-TS
placental								
10%	0.38	0.52	0.54	0.73	0.49	0.43	0.92	**0.94**
25%	0.03	0.07	0.07	0.28	0.08	0.04	0.52	**0.63**
50%	0.00	0.00	0.00	0.02	0.00	0.00	0.04	**0.09**
75%	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
hunting dog								
10%	0.69	0.83	0.81	0.91	0.80	0.68	0.98	**1.00**
25%	0.23	0.39	0.35	0.70	0.37	0.14	0.83	**0.94**
50%	0.00	0.02	0.02	0.19	0.02	0.00	0.25	**0.33**
75%	0.00	0.00	0.00	**0.01**	0.00	0.00	**0.01**	**0.01**
bird								
10%	0.57	0.75	0.75	0.96	0.70	0.76	0.97	**0.99**
25%	0.15	0.29	0.29	0.85	0.24	0.31	0.73	**0.87**
50%	0.01	0.02	0.02	**0.44**	0.02	0.02	0.09	0.23
75%	0.00	0.00	0.00	**0.09**	0.00	0.00	0.00	0.00
arthropod								
10%	0.59	0.77	0.75	0.94	0.74	0.85	0.96	**1.00**
25%	0.17	0.36	0.37	0.81	0.33	0.52	0.87	**0.91**
50%	0.01	0.05	0.05	**0.52**	0.05	0.11	0.32	0.49
75%	0.00	0.00	0.00	**0.14**	0.00	0.01	0.02	0.06
reptile								
10%	0.75	0.85	0.83	0.92	0.85	0.89	0.95	**0.98**
25%	0.33	0.55	0.52	0.78	0.55	0.58	0.88	**0.89**
50%	0.04	0.15	0.13	0.49	0.12	0.13	**0.57**	0.53
75%	0.00	0.00	0.00	0.10	0.01	0.01	**0.11**	0.07



(a) Unknown Superclasses

(b) Unknown Subclasses

Figure 6.6: OSCR CURVES.

the weights of the classification layer of the next higher hierarchical level. Based on the reported finding, this seems to be an effective way to improve semantic conditioning, even for classes where none of the other networks showed any effective semantic conditioning. The semantic conditioning is comparable to that of the ASN for *living thing* subclasses, performing slightly better or worse. Unlike the ASN, they can effectively be conditioned for *artifact* subclasses. From the conducted experiments, it can also be deduced that this effect stems from explicitly providing the ground truth hierarchical structure during training, as the ASN-TS works the same way as the ASN and the ASN-T as the ASN-A during validation.

Regarding semantic conditioning, it thus seems best to rely on the ground truth class to select the weight vector $\vec{v}_b$ rather than the predicted one. As soon as uncertainty is involved, good performance is only achieved when summing up $\vec{v}_b$ using the probability distribution as the ASN and ASN-TS do. This finding is also supported by the fact that the ASN likely integrates information from the ground truth sub- and superclasses through its probability distribution and consequently has good semantic conditioning. ASN-A, on the other hand, greatly relies on the accuracy of the higher hierarchical prediction and thus might struggle to learn from only the predicted classes.

The ASN-TS, thus, is an architecture that learns the decision boundaries between target classes of different subclasses well and is also able to use the attention mechanism to achieve the best accuracy score on the lowest hierarchical level. This shows that the attention mechanism can be used to learn a given hierarchy.

6.5 Open Set

To conclude the experiments, the performance of the models is measured when confronted with unknown classes. For this, the CCR and FPR are plotted as OSCR curves in Figure 6.6. Two different sets of unknown classes are presented to a selection of models: Samples of unknown superclasses and samples of unknown subclasses. The baseline model is compared with the three models that possess the best semantic conditioning, the ASN, ASN-T, and ASN-TS.

As seen in Figure 6.6a, the ASN is much better at distinguishing known samples from samples of unknown superclasses. The ASN is the best of all five models at lower FPRs, with the other

models converging for higher FPRs. The ASN can consistently achieve a higher CCR with the same FPR than the other rates under different θ. Interestingly, only the curve of the ASN-TS does not extend to low FPR values. This indicates that when $\theta = 1$ is reached, there still is a high FPR.

From Figure 6.6b it is also apparent that all networks perform equally badly when confronted with classes from a known superclass but an unknown subclass. The networks all struggle to differentiate these samples from known samples. This is understandable to a certain degree, as they are closer semantically related to the classes they have been trained on. The networks likely struggle with the 60 dog classes in the unknown subclasses. These are semantically closely related to samples of the *hunting dog* subclass and thus likely are not rejected as unknowns. In Figure 6.6b only slight differences are visible, namely that the ASN achieves a slightly higher CCR than the other models. The curves also extend towards much lower FPR, indicating that even very high θ unknown samples are not rejected. The only exception is again the ASN-TS, which again does not extend to low FPR, indicating that the threshold is maximized at $\theta = 1$.

There exists the potential for the attention mechanism to be applied to open-set classification, as demonstrated by the ASN's OSCR curve for unknown superclasses. When confronted with classes from unknown subclasses but known superclasses, no substantial effect can be seen. Most notably, the ASN-TS, seemingly able to learn the decision in the closed-set setting effectively, does not extend to low FPR values, indicating that it is ineffective at rejecting unknown samples. A limiting factor for the attention mechanism of ASN-TS is possibly that the attention mechanism is not designed to handle unknown classes. A feature transformation vector $\vec{v}_b$ is calculated and applied to the lower hierarchical level even when the confidence score for a super- or subclass is low. This means that the transformed features given to the target classifier are transformed even if the maximal confidence for a given subclass is low. However, it is unclear why this effect can be exclusively observed for the ASN-TS, and not for the ASN-T, which is trained in the same way.

Chapter 7

Discussion

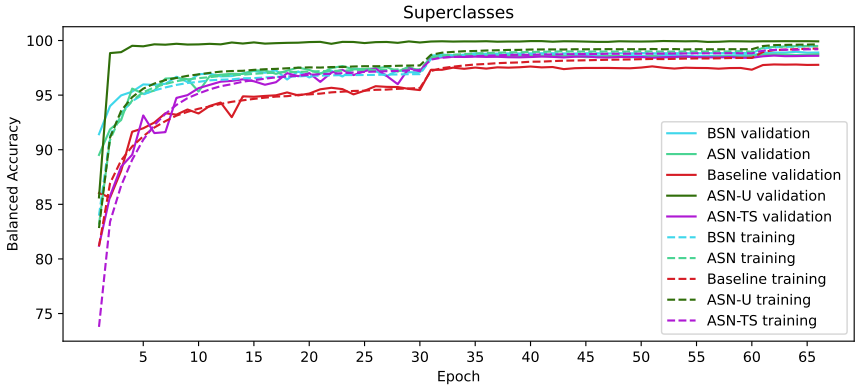
Based on the experiments and results presented in Chapter 6, various points of interest are discussed in this chapter.

7.1 Training of The Networks

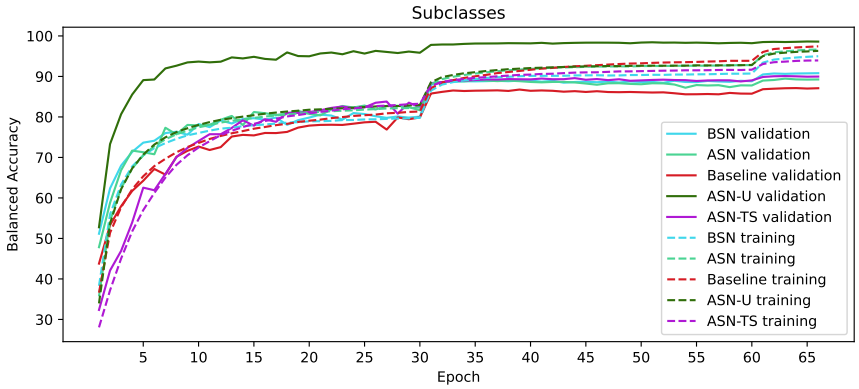
7.1.1 Overfitting

During training, some interesting behavior can be observed regarding the difference between the training and validation balanced accuracy scores. A model achieving a considerably lower score on the validation set is an indication that the model does not generalize well to unseen data. This behavior is commonly called overfitting, as the model fits too closely to the data points it has seen. To a certain extent, this behavior is to be expected since the model knows the ground truth labels of the training images. Still, overfitting is an undesirable trait due to the aforementioned lack of generalizability. To analyze this, the training and validation balanced accuracy scores for each epoch are plotted in Figure 7.1. The models ASN-A, LSN, and ASN-T are excluded from this analysis, as these networks are less important for the findings of this thesis. In Figure 7.1(a), the balanced accuracy scores of the superclasses are shown. On this hierarchical level, the difference in performance between validation and training accuracy is very low. The only notable occurrence is the ASN-U, which performs *much* better than the other model on the validation set during early epochs. The validation balanced accuracy score is also clearly better than the training one, which only converges during later epochs. For the subclasses, as depicted in Figure 7.1(b), the same pattern emerges for the ASN-U. Here, there are also visible differences between the other models, such as the baseline model reaching the lowest validation score and the highest training score after 66 epochs, indicating that it is the most prone to overfitting on this hierarchical level. In the target classes, as seen in Figure 7.1(c) the ASN-U shows no unordinary behavior. On this hierarchical level, the compared models all reach similar accuracy scores on the validation set. The training scores of the models, however, cover a range of values after 66 epochs. The ASN has the highest training score, indicating that this model may be prone to overfitting. The ASN-TS also reaches a fairly high training accuracy. It also possesses the highest validation accuracy, however. Interestingly, the BSN has the lowest training accuracy while having the second highest accuracy. This may indicate that the attention mechanism results in more overfitting compared to the (relatively simple) BSN.

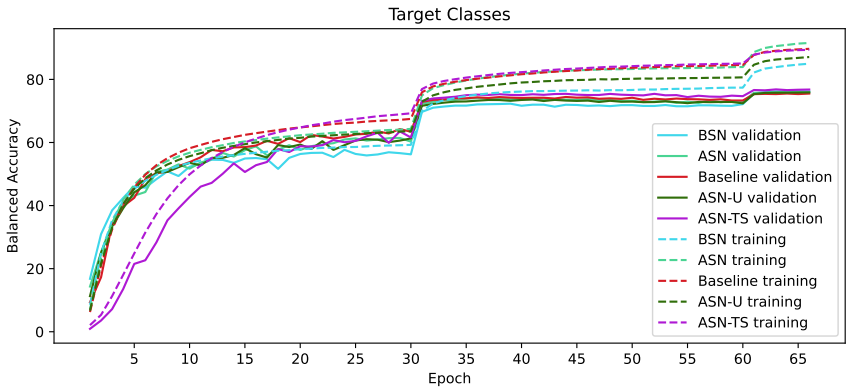
In all graphs in Figure 7.1, it can be seen that there is a significant increase in accuracy at the 30th and 60th epoch because of the StepLR, which is set to be increased at these points. Furthermore, the training of all networks could likely be shortened, considering that the validation scores stagnate or decrease on the interval between epochs 30 and 60. Such shortened training



(a) Superclass training and validation balanced accuracy



(b) Subclass training and validation balanced accuracy



(c) Target class training and validation balanced accuracy

Figure 7.1: OVERFITTING OF BASELINE, BSN, ASN, ASN-U, AND ASN-TS. *The training and validation balanced accuracy are plotted for each of the 66 epochs.*

could be done via hyperparameter tuning, such as the StepLR, or by exchanging the SGD optimizer through another one and would likely result in less over-fitted models.

The large discrepancy between the training and validation balanced accuracy of the ASN-U is puzzling. Possible explanations include an error when calculating the balanced accuracy, data leakage, or a bias in the validation set. However, data leakage and incorrect calculations are unlikely, as all models implement the same training and validation loop. Consequently, erroneous code would have influenced other models as well. One explanation is that the model performs worse during training due to the random crop and horizontal flip applied to the training images, whereas the validation images are not augmented, and the center crop is taken, possibly leading to better performance for the unaugmented validation images. Still, this does not explain why such an effect is not visible for different models. For the same reason, a bias in the validation set also is unlike, as this again would affect other models. It is not apparent what the root cause of the increased performance on the validation split is. While the ASN-U shows large promises for correct classification of higher-level, it fails regarding the semantic conditioning of the target classes. This is expected, as no attention mechanism is implemented for the target class prediction. Overall, the results of the ASN-U model should be enjoyed with caution, as the difference between validation and training sets raises some unanswered questions.

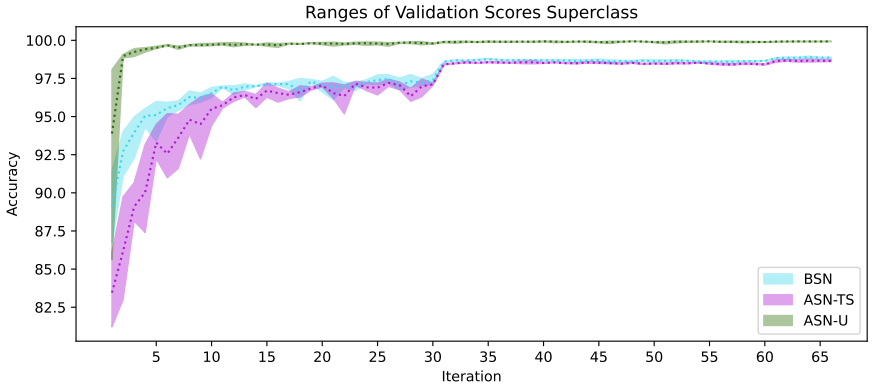
7.1.2 Robustness of Results

Training neural networks is an inherently stochastic process. In this thesis, several components use randomness. For example, randomness is employed for initializing the weights, the ordering of samples in each epoch, or during data augmentation. For this reason, it may be possible that parts of the observed differences are due to randomness. To observe this effect, the BSN, ASN, and ASN-U are each trained two more times to observe the differences. This means that now there are three balanced accuracy scores on the validation set in. In Figure 7.2, the average, maximum, and minimum of the three runs are plotted for each model. As can be seen in Figure 7.2(a) and Figure 7.2(b), the ASN-U achieves consistently good results, indicating that the high accuracy on the validation set also cannot be explained through sheer chance. Generally, the variation in balanced accuracy for all three models is high in the beginning and decreases strongly in later periods. In Figure 7.2(c), the most notable difference is visible in later periods, especially for the ASN-TS and BSN. The lowest accuracy of the ASN-TS is still higher than the highest of the BSN, indicating that the results described in Chapter 6 cannot be explained through stochasticity during training.

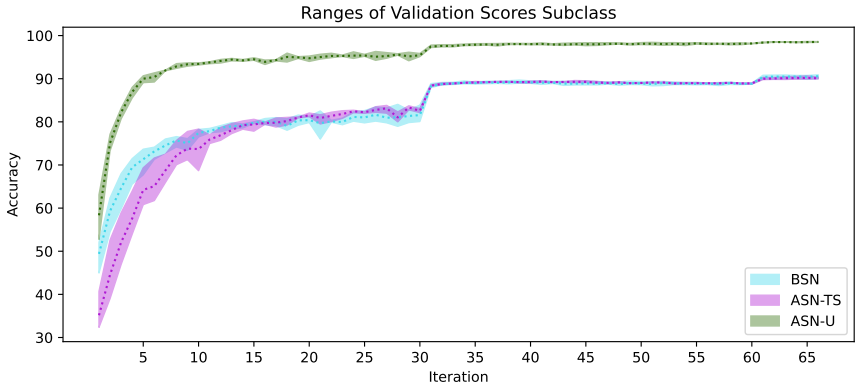
7.1.3 Hyperparameter selection

As the goal of this thesis was not the highest possible performance on the target class classification, the hyperparameters for training are selected so that a decent performance of the baseline network is achieved. The introduction of networks that share a backbone brings a new hyperparameter. As the loss function is shared across all levels, it must be backpropagated jointly. In the experiments, each loss of a hierarchical level was given a weight λ_b. The superclass and subclass loss were weighted with 25% each, and the target loss with 50% tuning this hyperparameter most likely impacted the performance – or at least the convergence speed – on each level. These weights are selected arbitrarily; it is not even necessary that they sum up to 100%. Fine-tuning this loss weighting may improve the performance, but unfortunately is out of the scope of this thesis.

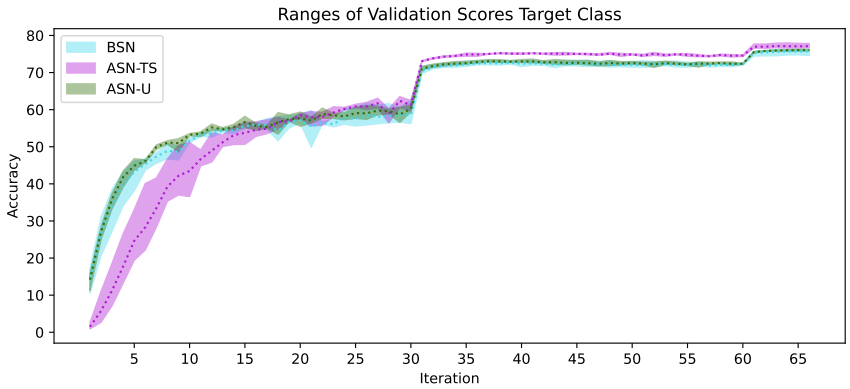
Generally, it seems reasonable to assume that some of the reached balanced accuracy of the models in this thesis may look different under different hyperparameters. Furthermore, it is also possible that each architecture variation has its own best set of hyperparameters. However, this



(a) Superclass training and validation balanced accuracy



(b) Subclass training and validation balanced accuracy



(c) Target class training and validation balanced accuracy

Figure 7.2: RANGE OF VALIDATION SCORES. *The range of balanced accuracy scores of three runs from each model is plotted as a colored area. The dotted line shows the average of the three runs.*

would make the comparison between models impossible, and as such, the approach taken is justified.

7.2 Visual and Semantic Hierarchy

One interesting observation is that even a baseline network makes mistakes inside of the semantic hierarchy despite it having *no* knowledge of the other levels of the hierarchy. Especially for *living things* superclass, mistakes are made close to the diagonal in the confusion matrix. Bilal et al. (2017) has already remarked on this phenomenon, whose approach was building a hierarchy based on confusion matrices. This misclassification pattern likely stems from the visual similarities of the images. Consequently, the difference and limitations of semantic hierarchies must be acknowledged.

The *living things* subclass samples likely have a greater intra-class (visual) similarity than those of the *artifact* superclass. Qualitative analysis also brings evidence of this. For example, in the *device* subclass, there can be found great variation between target classes. This contains classes such as *jack-o'-lantern*, *crane*, and *assault rifle*. While these three objects certainly can be described as devices, they do not share much visual similarity. There certainly is *some* visual similarity, especially regarding low-level features such as edges. The larger issue at hand is most likely that the visual similarity between samples of the same subclass is not high enough to distinguish them from samples of a different subclass properly. For example, a *spatula* belongs to the subclass *implement*, but the visual similarity between a *spatula* and an *assault rifle* is comparable to the visual similarity between an *assault rifle* and a *crane*. It is also questionable how semantically related these classes truly are. For comparison, take the subclass *hunting dog*. A *Norwich Terrier* and a *Norfolk Terrier* look nearly identical to the untrained eye.

One effect where this can be seen is with the attention mechanism of the ASN. This attention seems to be only successful in increasing the semantic conditioning of a network if the samples of a subclass share a high intra-class visual similarity. This indicates that the chosen hierarchy limits how well the attention mechanism works. The hierarchy of organisms can be structured according to their biological taxonomy, which often corresponds to a high visual similarity. For non-organic entities, the semantic concepts do not necessarily imply visual similarity. Concepts such as *device*, *implement*, or *covering* describe how entities are used rather than make any statements of their physical appearance.

This thesis delivers a concrete approach to teaching networks the decision boundaries between semantic unrelated classes, even if these classes are not distinct in their appearance from each other. As seen by the semantic conditioning of the ASN-T and ASN-TS, providing the ground truth labels of a higher hierarchical level can be effectively integrated into the attention mechanism, allowing the network to distinguish target classes from semantically different but visually similar subclasses from each other.

7.3 Semantic Conditioning Of Superclasses

In Chapter 6, only the semantic conditioning between the target and subclass is considered, following the definition in Section 5.3.2. In (5.14) semantic conditioning is only defined between a parent and a child class. However, by replacing the function $\mathrm{parent}(c)$ with the function $\mathrm{grandparent}(c)$, one can get the grandparent class of a given class c. Concretely, one can measure how well a target class is semantically conditioned towards its superclass. The CCDF of the target-superclass semantic conditioning is depicted in Figure 7.3 and quantified in Table 7.1.

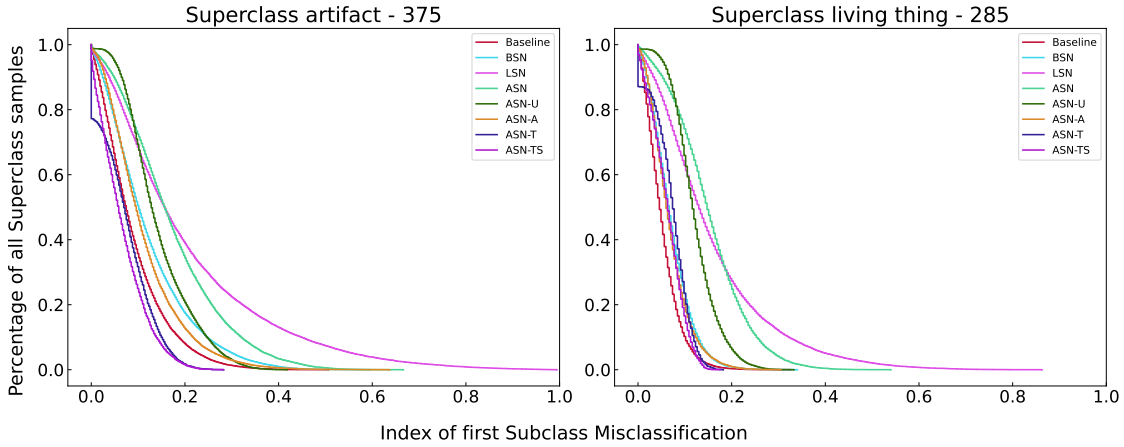


Figure 7.3: SUPERCLASS HISTOGRAMS.

Table 7.1: .

Percentage	Baseline	BSN	LSN	ASN	ASN-U	ASN-A	ASN-T	ASN-TS
artifact								
10%	0.36	0.51	0.65	**0.73**	0.69	0.47	0.32	0.25
25%	0.03	0.10	**0.27**	0.22	0.10	0.06	0.00	0.00
50%	0.00	0.00	**0.06**	0.01	0.00	0.00	0.00	0.00
75%	0.00	0.00	**0.01**	0.00	0.00	0.00	0.00	0.00
living thing								
10%	0.10	0.24	0.66	**0.74**	0.66	0.20	0.23	0.17
25%	0.00	0.00	**0.20**	0.11	0.01	0.00	0.00	0.00
50%	0.00	0.00	**0.02**	0.00	0.00	0.00	0.00	0.00
75%	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

The architectures that show the best semantic conditioning for subclasses – ASN-T, and ASN-TS – are surprisingly weakly conditioned toward the superclasses. This implies that these two networks do not properly learn the decision boundary between the two superclasses. The best of the proposed architecture is the LSN, which showed little semantic conditioning for the subclasses. This is an indication that the LSN mechanism still has potential and may be able to deliver better results through further modifications. Interestingly, the ASN can still achieve the second-best results. This indicates that the issue does not lie with the attention mechanism in general. It is more likely that the low performance of ASN-T and ASN-TS is caused by the ground truth labels provided during training. One explanation could be that these two architectures focus entirely on learning the decision boundaries between subclasses through the transformed features $\vec{\varphi}'_3$. These transformed features are only calculated using information from the subclass classifier through the transformation vector $\vec{v}_2$, without directly incorporating information from the superclasses. This enables them to learn the semantic relationship of classes that belong to subclasses that are not visually distinct from each other. However, the trade-off that may occur is that these models rely less on the visual similarity of the input image. A possible remedy to increase the semantic conditioning of the target classes toward the superclass is to incorporate superclass information into the calculation of $\vec{\varphi}'_3$. A straightforward approach that might work is to create a weighted average of all previous $\vec{v}_b$ in (5.6).

7.4 Open-Set Architecture

From the proposed architectures, only the original ASN architecture achieves distinctly better performance than the other architectures. This finding is also limited to classes from unknown superclasses. The failure to reject samples from known superclasses but unknown subclasses is somewhat understandable, as these classes are not truly unknown. Since all architectures, except for the baseline, share a ResNet backbone, it is not a surprise that they fail to properly reject these classes.

As seen by the ASN, the attention mechanism may have potential use cases for open-set classification. However, there likely is much potential to increase the performance for open-set classification. A common strategy is to incorporate negative samples during training (Palechor et al., 2023), in the hopes of learning the decision boundary between known and unknown classes. Such an approach may already greatly increase the open-set performance of ASN-TS, as it heavily relies on ground truth labels during training. By providing samples of unknown classes, the attention mechanism may learn a feature transformation vector v_b to distinguish known from unknown classes.

Still, negative samples cannot possibly cover the space of the unknown classes. As such, the attention mechanism should also be able to handle unknown classes explicitly. Generally, v_b could be changed in two ways to handle uncertainty for the next lower level. Via negative samples, there could be a garbage transformation vector. A common open-set approach uses a garbage - or background - class Dhamija et al. (2020), which is trained with the negative samples. With such a garbage class, the transformation vector v_b would consider whether the class predicted for b is an unknown sample and use this information for lower hierarchical level $b + 1$.

An alternative way is to threshold the sharing mechanism based on the maximum probabilities predicted for level b. The attention mechanism, thus, would only be applied when the network is certain in its higher-level prediction. This gating mechanism would likely decrease the FPR and help to reject unknown samples as such, as it is likely that the feature transformation causes some confusion. However, this may impact the performance of the closed set. Consequently, the adaption of the attention mechanism lies beyond the scope of this thesis and will be left to future work.

7.5 Points of Failure

Qualitative analysis can help to gain a deeper understanding of where the network struggles. As a start, misclassifications made by the original ASN are analyzed. However, due to the size of ImageNet, the analysis is limited to the highest hierarchical level, as the accuracy is the highest. The ASN reaches a balanced accuracy of 98.75% for superclasses. Out of the 33'000 validation images, 404 images were misclassified at that level. This makes it feasible to explore the causes of misclassification. One thing that becomes quickly apparent is (semantically) overlapping samples. In Figure 7.4, two images depict a dog with a tennis ball. The validation crop of the image is displayed, so the images are depicted the way they were presented to the network. However, one of the images is labeled as a dog, while the other is labeled as a tennis ball. These are samples from two semantically distant classes, as the tennis ball belongs to the *artifact* superclass, and the dogs belong to classes from the *living thing* superclasses. Similarly, in Figure 7.5, there are two images of a spider in its web. In both examples, the model made a perfectly reasonable prediction: The image contains a class known to the network, and thus, it predicts that class. In the case of the spider, the network even predicts the correct target class: a black and gold garden spider. Since each picture only has one ground truth label, such predictions are considered incorrect despite being very reasonable. While the synsets of the ImageNet data set are not overlapping, the

(a) Ground truth: Artifact - Equipment - Tennis Ball
Predicted: Living Thing - Hunting Dog - English Springer

(b) Ground truth: Living Thing - Hunting Dog - Yorkshire Terrier
Predicted: Artifact - Equipment - Tennisball

Figure 7.4: DOGS AND TENNISBALLS. *In Figure 7.4(a) the image is labeled as a tennis ball, while in Figure 7.4(b) the image is labeled as a dog.*

(a) Ground truth: Artifact - Device - Spider Web
Predicted: Living Thing - Arthropod - Black and Gold Garden Spider

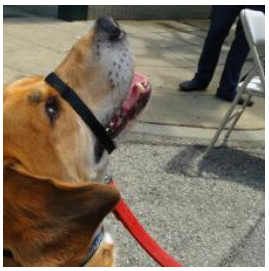
(b) Ground truth: Living Thing - Arthropod - Barn Spider
Predicted: Artifact - Device - Spider Web

Figure 7.5: SPIDERS IN THEIR WEBS. *Both subfigures depict a spider in their spider web. However, Figure 7.5(a) is labeled as a spider web and Figure 7.5(b) as a spider.*

(a) Ground truth: Artifact - Clothing - Shower Cap
Predicted: Living Thing - Clothing - Shower Cap

(b) Ground truth: Artifact - Covering - Nipple
Predicted: Living Thing - Covering - Nipple

(c) Ground truth: Artifact - Device - Loupe
Predicted: Living Thing - Device - Loupe



(d) Ground truth: Artifact - Device - Muzzle
Predicted: Living Thing - Device - Muzzle

Figure 7.6: ONLY INCORRECT SUPERCLASSES. *All for images are classified as the correct sub- and target class, but as the incorrect superclass.*

(a) Ground truth: Artifact - Device - Knot
Predicted: Living Thing - Arthropod - Crayfish

(b) Ground truth: Artifact - Device - Microphone
Predicted: Living Thing - Reptile - Terrapin

(c) Ground truth: Artifact - Device - Nail
Predicted: Living Thing - Reptile - Water Snake

(d) Ground truth: Artifact - Structure - Restaurant
Predicted: Living Thing - Arthropod - Dungeness Crab

Figure 7.7: SEVER MISTAKES - ASN-U. *These images were misclassified across all hierarchical levels by ASN-U.*

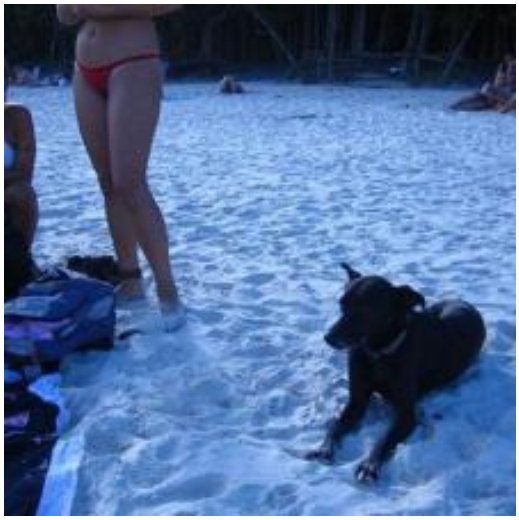
content of the samples may evidently be. However, such limitations should be expected when dealing with single labels in large-scale data sets.

In any images in the wild, i.e., not in a studio setting, more than just a single subject exist. This, of course, becomes problematic for classification, especially if there are other subjects in the pictures which a model knows and recognizes. In the examples in Figure 7.4 and Figure 7.5, the classes were confused for each other. This means that the prediction is consistent, i.e., the prediction itself applies if the target class label is that of a different object. However, often only parts of the hierarchy are incorrectly predicted. This is due to the fact that there are no constraints on what kind of predictions the network can make. In Figure 7.6, examples can be found where only the superclass is predicted incorrectly. This means that the classifiers of the two lower levels made completely accurate predictions, but that of the superclass focused on the wrong part of the picture.

Integrating lower-class information into higher-class predictions delivers a plausible explanation of why the ASN-U may be this good at classifying the super- and subclasses compared to the ASN architecture. Reaching a balanced accuracy of 99.92% for superclasses, only 48 samples are misclassified. In Figure 7.7, mistakes from ASN-U are depicted. If the target class has been correctly predicted - or even if a related class is predicted - the prediction for the higher hierarchi-

cal level should be much easier. In other words, the lower hierarchical class is a strong indicator of the higher hierarchical class rather than the other way around. However, the network sometimes still fails severely. A severe misclassification in this scenario is a misclassification where the network predicts (parts) of the hierarchy which are not contained in the picture. Examples of such misclassification made by ASN-U are depicted in Figure 7.7. Here again, it should be noted that in all four examples, hierarchically sound predictions are made by ASN-U. Still, these are examples of severe mistakes that should be avoided as much as possible. However, it should also be mentioned that at least some of the examples exhibit low-label quality, such as Figure 7.7(b) where a tattoo of a microphone is labeled as a *microphone* and thus a *device* and *artifact*. This sample technically represents an open-set issue – as *tattoo* is not a class contained in ImageNet-1k – highlighting the usefulness of open-set consideration. While this is a low-quality label, the mistake made by the model is still severe, as it predicts a *terrapin* which is a turtle species. Similarly, Figure 7.7(d) mistakes a *restaurant* for a *dungeness crab*, likely because the image is a close-up shot of a meal rather than the restaurant itself. Still, while low-quality labels are likely responsible for a formidable part of the mistakes, so definitely stem from the network, such as those depicted in Figure 7.7(a) and Figure 7.7(c) where inanimate objects are mistaken for animals.

Now, as the final point of discussion, the differences between predictions from the ASN and ASN-TS for the same image are analyzed. Again some interesting samples are manually selected for discussion. These examples fulfill two conditions. First, as in the other examples, they stem from predictions where both models incorrectly classified the superclass to allow for manual analysis. Secondly, the predictions from both models differ. The examples are shown in Figure 7.8. Here the effect of the training with ground truth labels for the attention mechanism becomes apparent. In Figure 7.8(a), both ASN and ASN-TS predict *living thing* as a superclass, most likely as a dog is prominently displayed in the image. For the subclass, both of them correctly predict *clothing*. While the ASN-TS is effectively able to follow up with a target class belonging to the correct subclass *swimming trunk*, the ASN instead predicts a dog breed. Similarly, in Figure 7.8(b), ASN-TS correctly predicts the correct subclass and follows up with the correct target class. In contrast, ASN indeed manages to correctly predict the correct target class *cowboy hat*. However, this target class does not belong to the predicted subclass *reptile*. In Figure 7.8(c), the ASN makes widely inaccurate predictions on each hierarchical level, while the ASN-TS can identify the correct subclass *bird*. The ASN-TS predicts an incorrect target class *kite*, which is closely related to the ground truth class *cock* as both belong to the subclass *bird*. Finally, in Figure 7.8(d), both networks correctly identify the correct subclass. However, the ASN then predicts *chainlink fence* as a target class, which is not related to the predicted subclass. This is, of course, understandable, as a *chainlink fence* is visible in the image. The ASN-TS can integrate the information from the subclass more effectively and thus correctly predicts the true label *dingo*. These examples show how the ASN-TS can often leverage information gained from the subclass classification and make a semantically consistent prediction. The effect displayed is limited to the subclass-target relationship, as only images from incorrectly predicted superclasses are analyzed. Consequently, the relationship between super- and subclasses cannot be seen.



(a) Ground truth: Artifact - Clothing - Bikini
ASN-TS: Living Thing - Clothing - Swimming Trunks
ASN: Living Thing - Clothing - Labrador Retriever

(b) Ground truth: Artifact - Clothing - Cowboy Hat
ASN-TS: Living Thing - Clothing - Cowboy Hat
ASN: Living Thing - Reptile - Cowboy Hat

(c) Ground truth: Living Thing - Bird - Cock
ASN-TS: Artifact - Bird - Kite
ASN: Artifact - Placental - Breakwater

(d) Ground truth: Living Thing - Placental - Dingo
ASN-TS: Artifact - Placental - Dingo
ASN: Artifact - Placental - Chainlink Fence

Figure 7.8: SAME PICTURE - DIFFERENT PREDICTIONS. *This figure contains images for which both the ASN and ASN-TS incorrectly predicted the superclass.*

Chapter 8

Conclusion

8.1 Summary

This thesis explored several architectures that employ a hierarchical classification to learn the decision boundaries between semantically related classes. A ground-truth hierarchy was built that contains 660 of the ImageNet-1k classes. This hierarchy comprises two superclasses, twelve subclasses, and the 660 target classes. To evaluate how the models perform, two main evaluation metrics were used. The simpler measurement of these two is the balanced accuracy scores calculated for each hierarchical level. Balanced accuracy was used to account for imbalanced sub- and superclasses. The second is semantic conditioning, expressed through a CCDF that shows how at which indices the networks predict target classes that do not belong to the same subclass as the ground truth sample.

In the first experiment, a baseline model was compared to the BSN architecture to see whether a model with a shared ResNet-49 backbone outperforms a model consisting of three separate ResNet-50 networks. The BSN architecture achieved better-balanced accuracy on each hierarchical level, showing that a single multi-task network outperforms multiple single-task networks.

In the next experiments, it was explored how the BSN can be extended so that information between the different hierarchical classifiers can be shared. The result of these extensions is the LSN and ASN. The former appends the predicted logits of the higher-hierarchical level and achieves decent balanced accuracy scores but low semantic conditioning from the target classes towards the subclasses. The latter implements an attention mechanism that creates a transformation vector from the weights of the higher-level classifier. This transformation vector transforms the deep features to incorporate information from the higher hierarchical level to the next lower one. The attention mechanism of the ASN had partial success in semantically conditioning the target classes towards both the sub- and superclasses. Still, the LSN successfully conditioned the target class towards the superclass. The attention mechanism of the ASN was further fine-tuned with several more proposed architectures.

The ASN-TS and ASN-T architecture show that incorporating ground-truth information of the hierarchy during training greatly increases semantic conditioning, even for samples of subclasses that are not very visually distinct from each other. The ASN-TS also achieves the best accuracy score on the target classes, showing that its attention mechanism has a positive effect on classification accuracy at the lowest hierarchical level. On the higher hierarchical levels, the ASN-U achieves the best classification accuracy by reversing the attention mechanism so that information from the lower level is used for predictions on the higher level. However, the results of ASN-U leave some unanswered questions, namely that it performs much better on the validation than on the training set. These results should thus be enjoyed with caution.

These models were also tested in an open-set setting where the remaining 340 classes were

split into two sets. One set is made of classes from unknown superclasses, the other from unknown subclasses but known superclasses. The only model that possesses a good performance is the ASN, by achieving a higher CCR with the same FPR as other models. However, a clear difference is only visible for samples from unknown superclasses, as all networks struggled to reject samples from unknown subclasses. This attention mechanism consequently may have applications as an open-set classifier. Still, there would be the need to further modify the attention mechanism to be adapted for open-set classification.

Conclusively, the ASN-TS is an architecture that enables a model to learn the decision boundaries between subclasses well and achieves the best target class accuracy. However, its specialized training makes it worse at recognizing the decision boundaries between superclasses and decreases its open-set performance. In both these tasks, the ASN performs well. This thesis also suggests ways how the attention mechanism could be modified to improve performance on these tasks.

Overall, the introduced attention mechanism and its modifications enable neural networks to learn the semantic relationship between classes without introducing additional parameters that need to be trained. The various modifications show that this attention mechanism can be further adapted to excel at specific metrics, which may vary depending on the task at hand. This thesis presents an effective solution through architecture on how a network can learn the semantic relationship of a given set of classes.

8.2 Limitations

Despite the analysis performed in this thesis, some limitations naturally emerge and need to be considered for an accurate interpretation of the findings. The standard practice for supervised learning is to partition the data into three parts: training, validation, and test. The training set is employed to teach the model, the validation set is used for model performance evaluation and potential tuning, and the test set provides an unbiased evaluation of the final model's capacity to generalize to completely new data. In this thesis, the data has only been divided into a training set and a validation set. This means that the models have not been subjected to a final assessment using an entirely unseen test set. Consequently, the presented evaluations may not fully reflect the models' performance when exposed to completely new data.

It is worth noting that in this work, hyperparameters were not extensively tuned. Rather, a set of hyperparameters that demonstrated adequate performance for the baseline model were selected and were kept constant while creating and evaluating new architectures. The only exception is the loss-weight for each hierarchical branch λ_b. These weights were selected before any training run and not modified further. This approach somewhat mitigates the overfitting risk that can occur when hyperparameters are excessively tuned to improve performance on the validation set. However, the use of a single validation set both for evaluation and potential minor adjustments might lead to an overestimated view of the model's performance. This limitation could have been overcome by using a separate test set.

A notable exception to this limitation is the evaluation of the models on two open-sets. Here, completely unknown samples were used and thus represented an unbiased estimation of the performance on unseen data. Despite this, a separate test set could have offered additional insights into the models' ability to generalize, and thus, future research might benefit from its inclusion.

A further limitation is that all these findings have only been validated for the (somewhat arbitrarily) constructed hierarchy. What hierarchy is used may have a large impact on performance, as seen in the difference in performance between classes of the *artifact* and *living thing* superclasses. However, some ground-truth hierarchy needs to be selected, and what defines a good hierarchy greatly depends on the specific task at hand.

8.3 Future Work

This thesis proposes a general architecture with a hierarchical attention mechanism. One concrete use case is presented for such a mechanism: Open-set classification. As only a very basic evaluation of the models was conducted, it would be of high interest to pursue this approach with modifications that adapt the architecture for open-set recognition.

One thing that is important to note about the attention mechanism in this work is the fact that it does not introduce any extra parameters to train. This enables a fair comparison between models, as all models in this thesis – except the baseline and LSN – have the same number of parameters. Still, the attention mechanism could be expanded to contain one (or multiple) FC layer(s) instead of simple element-wise multiplication. This likely would improve the performance further, allowing for an even more adaptive attention mechanism.

Furthermore, this thesis focuses on solving the problem of conditioning networks to predict semantically related classes through their architecture. This approach has been shown to be successful. As such, it might be highly interesting to modify the loss function used in these architectures. After all, the loss function directly dictates what objective is to be reached. All models use weighted CCE, a loss function that does not consider how semantically related two predicted classes are. Such a loss function - in conjunction with a hierarchical architecture - may be a very successful approach in semantically conditioning neural networks.

Appendix A

Attachments

A.1 Word Embeddings for Hierarchy

As discussed in Chapter 4, a key challenge is the construction of a hierarchy. One of the main issues of the `robustness` algorithm for constructing a hierarchy is that not all 1000 classes are contained in the ground-truth hierarchy. This section suggests an approach to how this could be achieved. Rather than completely relying on the WordNet hierarchy, another method is proposed for estimating the semantic similarity between labels. As all labels have a representation in language, natural language processing can be used in order to extract some semantic meaning from the label. This approach is based on the distributional hypothesis (Harris, 1954). The underlying idea is that words with similar meanings occur in a similar context. Based on this assumption, machine learning techniques can extract semantic encoding from a given word or sequence of words. The advantage of using such encodings is that their high-dimensional embeddings contain semantic information, as they are constructed from the context a given word occurs. There are various approaches to creating such embeddings. In recent years, transformers have proven capable of creating semantically-rich word embeddings. One of the most notable models in this field is BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019). BERT is especially practical as pre-trained models exist. Still, theoretically, any given large language model should be able to create such embeddings.

These embeddings can then be used to group labels that are semantically similar together based on their similarity in the embedded space. One possible approach is to use the ImageNet labels and embed them. More interestingly, parts - or the entire - WordNet hierarchy can be integrated into the embedding processes. Rather than just embedding the single final label, any given sequence of hierarchical layers can be integrated. This means it is possible to integrate (parts) of the WordNet hierarchy to provide more context for labels. The following example illustrates this: The word *boxer* refers both to a breed of dogs and a person that boxes. The cosine similarity between the embedding for *boxer* and a *ballplayer, baseball player* is 0.38, and the similarity between *boxer* and a *Saint Bernard, St Bernard* is 0.46. These relatively similar values show that the embeddings of the ImageNet label alone do not provide enough information to discriminate between some classes properly. Even a human has trouble determining whether a person or a dog is meant when only the word *boxer* is used. However, once information from the WordNet hierarchy is used, the distinction becomes much clearer. When the parent label is also considered, the sequence of words to embed becomes *boxer/working dog*, *ballplayer, baseball player/athlete, jock*, and *Saint Bernard, St Bernard/working dog*. This results in the similarity between *boxer* and a *ballplayer, baseball player* to decrease to 0.35 and between *boxer* and *Saint Bernard, St Bernard* to rise to 0.79. If one considers four levels instead of just two, these numbers change to 0.25 and 0.93. In this example, it is clear why the similarity increases: The same words have the same

embedding, i.e., their similarity is 1. So the more parents in the encoded hierarchy are shared, the more similar their embeddings are. This allows consideration of the WordNet hierarchy when labels are embedded. Such a similarity metric allows for the usage of clustering methods to create superclasses. The advantage of using clustering is its versatility: When only a predetermined hierarchy, such as WordNet, is used, one has to decide on an arbitrary way to make this hierarchy fit for a supervised learning task. This can be a challenging task, especially when the number of classes grows large. A (hierarchical) clustering approach allows a more flexible way to create and group all classes into a hierarchy. However, here again, one has to decide how many levels the constructed hierarchy should have and how many clusters there should be at each level. This, combined with the fact that this thesis used the classes not belonging to the hierarchy for open-set classification, led to the decision to use the hierarchy presented in Section 4.3.

A.2 Confusion Matrices

In the Figures A.1, A.2, A.3, A.4, A.5, A.6 , the confusion matrixes of all models are depicted. As discussed in Chapter 6, they look all very similar, especially for the *living things* superclass. Here the vast majority of misclassifications happen along the axis. For the *artifact* superclass, most models make mistakes inside the right superclass but not in the correct subclass. The lonely and most notable exception is the ASN-TS in Figure A.1. This model visibly makes more mistakes inside the correct subclass, indicating that this model learns the decision boundary between samples from different subclasses.

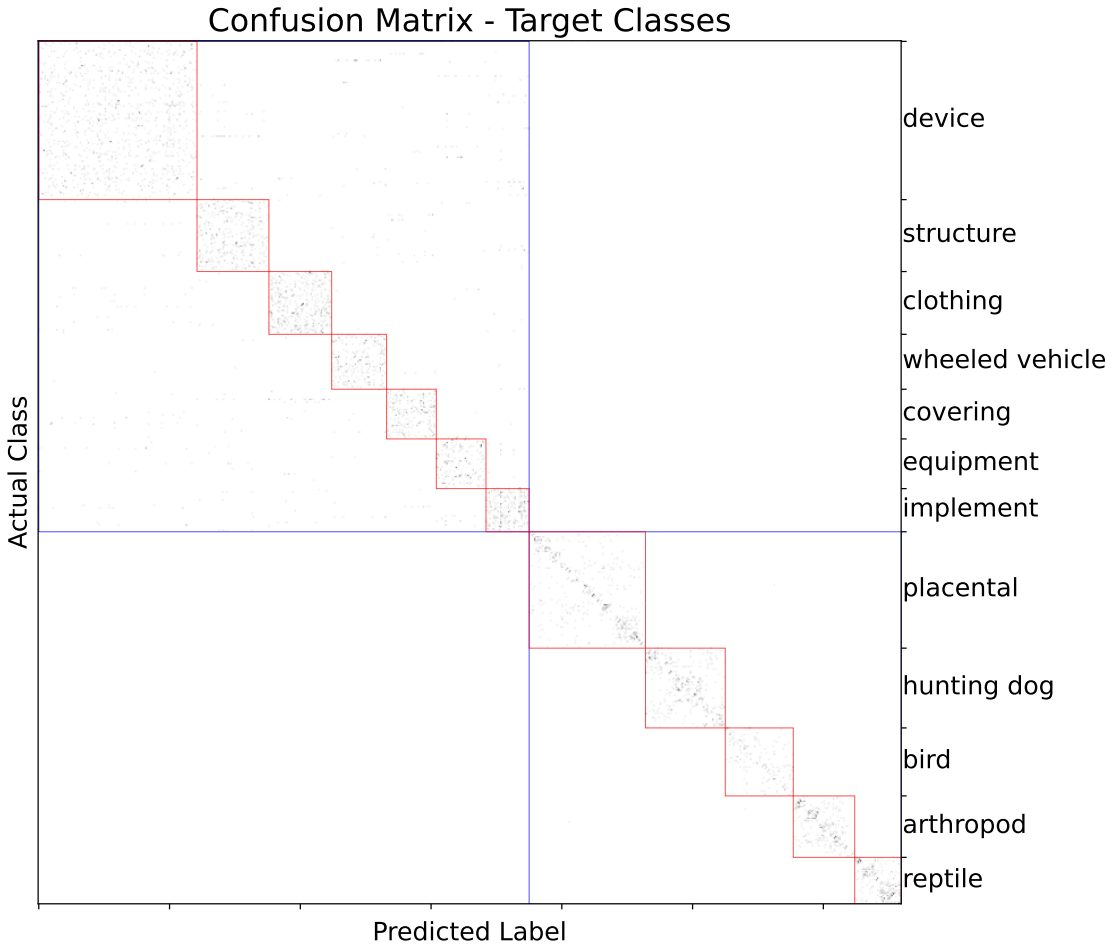


Figure A.1: ASN-TS CONFUSION MATRIX. *Misclassifications of ASN-TS in a log-scale.*

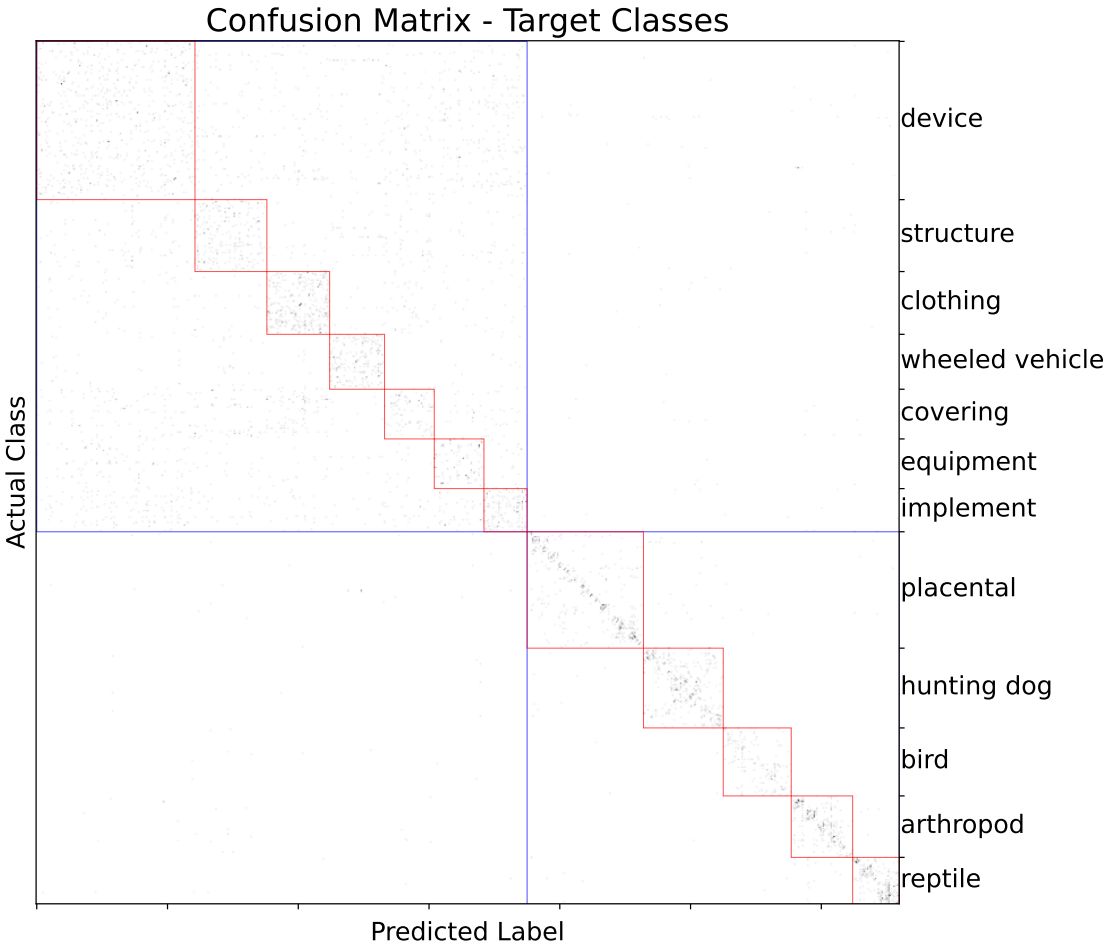


Figure A.2: ASN CONFUSION MATRIX. *Misclassifications of ASN, colored in a log-scale.*

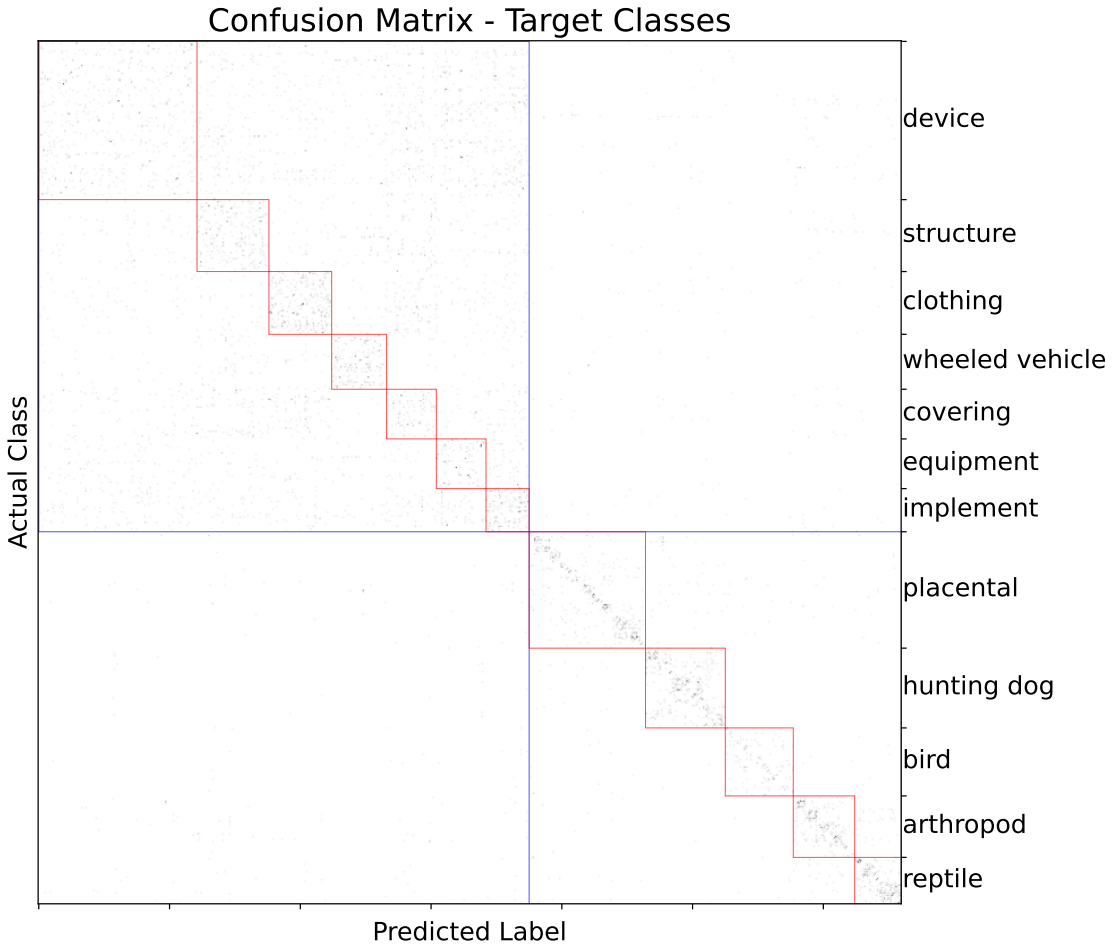


Figure A.3: ASN-T CONFUSION MATRIX. *Misclassifications of ASN-T, colored in a log-scale.*

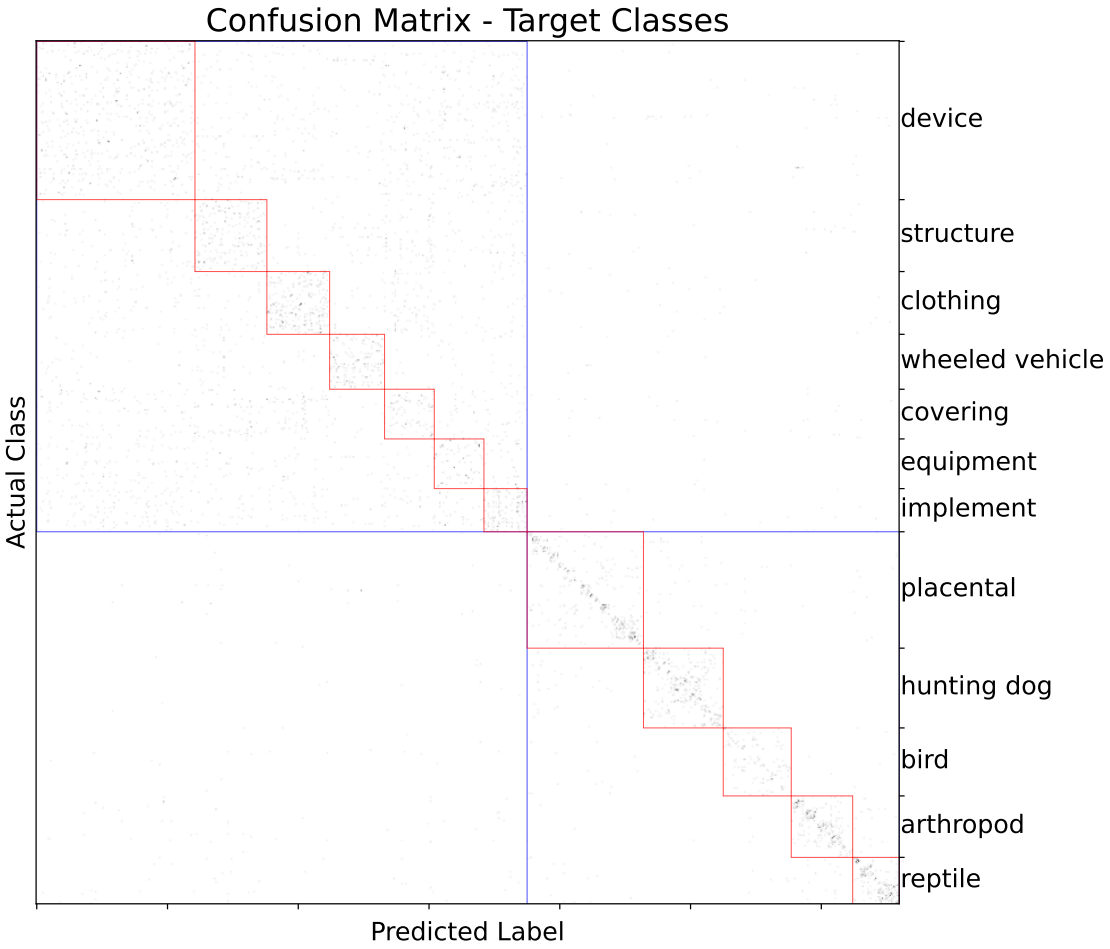


Figure A.4: ASN-U CONFUSION MATRIX. *Misclassifications of ASN-U, colored in a log-scale.*

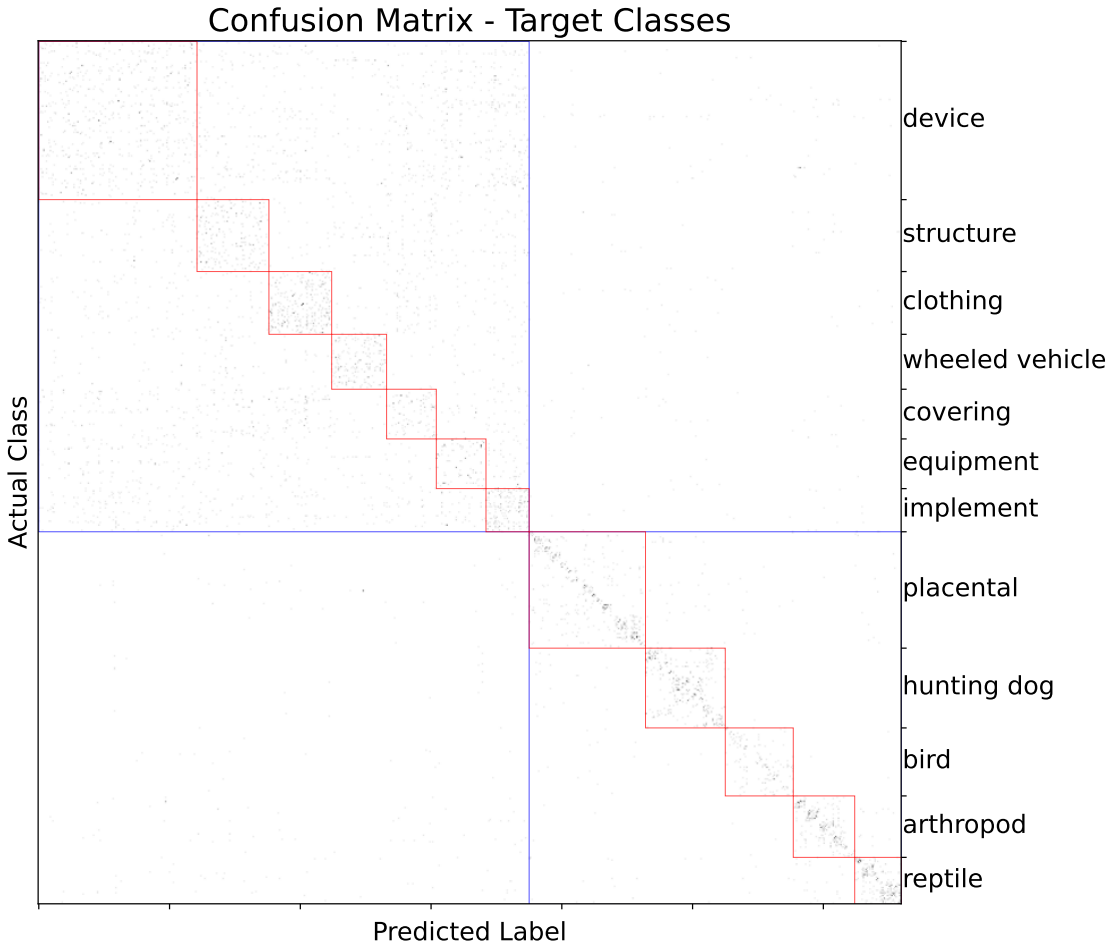


Figure A.5: ASN-A CONFUSION MATRIX. *Misclassifications of ASN-A, colored in a log-scale.*

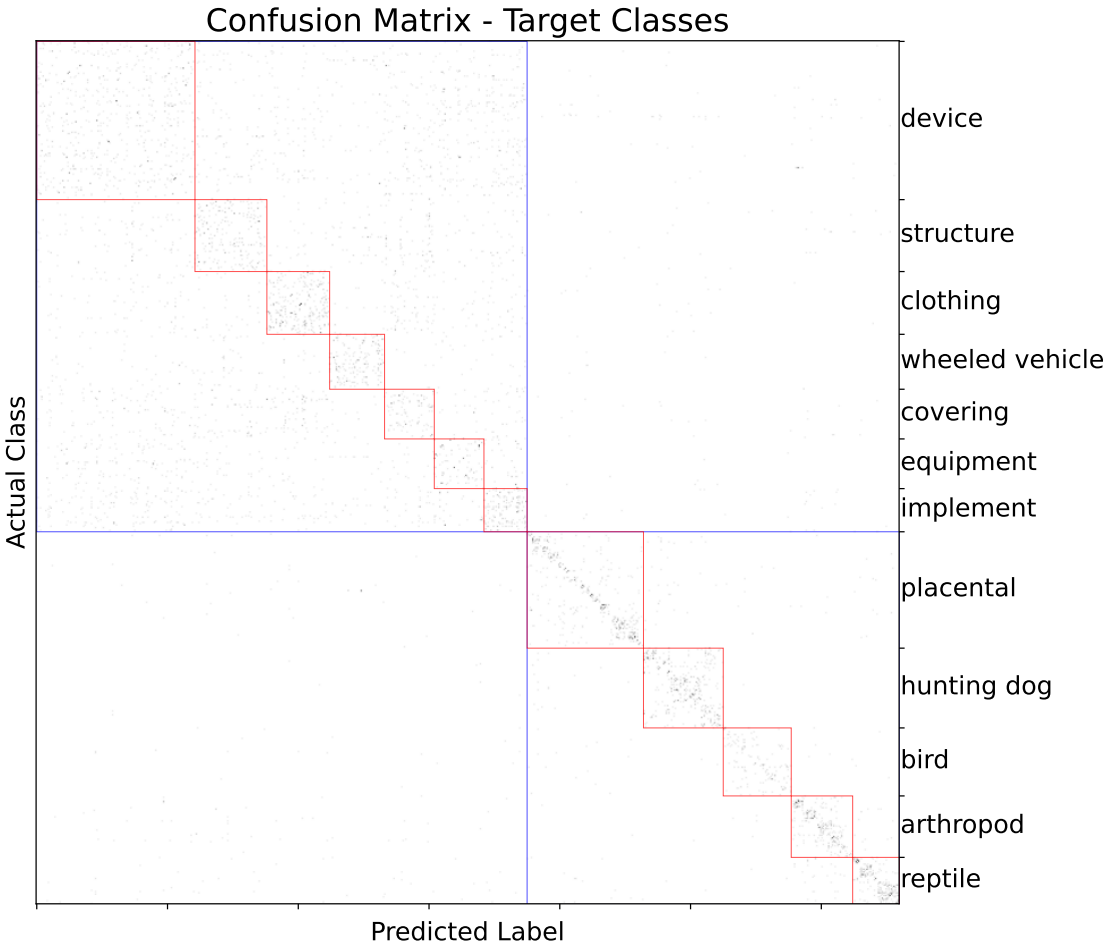


Figure A.6: LSN CONFUSION MATRIX. *Misclassifications of LSN, colored in a log-scale.*

List of Figures

List of Tables

Bibliography

Bendale, A. and Boult, T. E. (2016). Towards open set deep networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1563–1572.

Bertinetto, L., Mueller, R., Tertikas, K., Samangooei, S., and Lord, N. A. (2020). Making better mistakes: Leveraging class hierarchies with deep networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12506–12515.

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

Chen, G., Qiao, L., Shi, Y., Peng, P., Li, J., Huang, T., Pu, S., and Tian, Y. (2020). Learning open set network with discriminative reciprocal points. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 507–522. Springer.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dhamija, A. R., Günther, M., and Boult, T. (2018). Reducing network agnostophobia. *Advances in Neural Information Processing Systems*, 31.

Dhamija, A. R., Günther, M., Ventura, J., and Boult, T. E. (2020). The overlooked elephant of object detection: Open set. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1010–1019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Engstrom, L., Ilyas, A., Santurkar, S., and Tsipras, D. (2019). Robustness (python library). https://github.com/MadryLab/robustness.

Ge, Z., Demyanov, S., Chen, Z., and Garnavi, R. (2017). Generative openmax for multi-class open set classification. In *British Machine Vision Conference 2017*. British Machine Vision Association and Society for Pattern Recognition.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Guo, Y., Liu, Y., Bakker, E. M., Guo, Y., and Lew, M. S. (2018). CNN-RNN: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 77:10251–10271.

Harris, Z. S. (1954). Distributional structure. *WORD*, 10(2-3):146–162.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hendrycks, D. and Gimpel, K. (2016). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

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

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

Johnson, J. M. and Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54.

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

Kornblith, S., Shlens, J., and Le, Q. V. (2019). Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2661–2671.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Lee, K., Lee, K., Min, K., Zhang, Y., Shin, J., and Lee, H. (2018). Hierarchical novelty detection for visual object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1034–1042.

Li, J., Jin, K., Zhou, D., Kubota, N., and Ju, Z. (2020). Attention mechanism-based cnn for facial expression recognition. *Neurocomputing*, 411:340–350.

Li, K., Wang, N. Y., Yang, Y., and Wang, G. (2021). SGnet: A super-class guided network for image classification and object detection. In *2021 18th Conference on Robots and Vision (CRV)*, pages 127–134. IEEE.

Luccioni, A. S. and Rolnick, D. (2023). Bugs in the data: How imagenet misrepresents biodiversity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14382–14390.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

Northcutt, C. G., ChipBrain, M., Athalye, A., and Mueller, J. (2021). Pervasive label errors in test sets destabilize machine learning benchmarks. *stat*, 1050:1.

Palechor, A., Bhoumik, A., and Günther, M. (2023). Large-scale open-set classification protocols for imagenet. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 42–51.

Pan, X., Ge, C., Lu, R., Song, S., Chen, G., Huang, Z., and Huang, G. (2022). On the integration of self-attention and convolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 815–825.

Princeton University (2010). About wordnet. https://wordnet.princeton.edu/.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Redmon, J. and Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271.

Roy, D., Panda, P., and Roy, K. (2020). Tree-CNN: A hierarchical deep convolutional neural network for incremental learning. *Neural Networks*, 121:148–160.

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

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252.

Seo, Y. and shik Shin, K. (2019). Hierarchical convolutional neural networks for fashion image classification. *Expert Systems with Applications*, 116:328–339.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society.

Tian, C., Xu, Y., Li, Z., Zuo, W., Fei, L., and Liu, H. (2020). Attention-guided cnn for image denoising. *Neural Networks*, 124:117–129.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vaze, S., Han, K., Vedaldi, A., and Zisserman, A. (2021). Open-set recognition: A good closed-set classifier is all you need. In *International Conference on Learning Representations*.

Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

Wu, H., Merler, M., Uceda-Sosa, R., and Smith, J. R. (2016). Learning to make better mistakes: Semantics-aware visual food recognition. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, page 172–176, New York, NY, USA. Association for Computing Machinery.

Yan, Z., Zhang, H., Piramuthu, R., Jagadeesh, V., DeCoste, D., Di, W., and Yu, Y. (2015). HD-CNN: Hierarchical deep convolutional neural networks for large scale visual recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2740–2748.

Yang, K., Yau, J., Fei-Fei, L., Deng, J., and Russakovsky, O. (2021). Imagenet. https://image-net.org/. Accessed: 2023-07-17.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.

Zha, Z.-J., Hua, X.-S., Mei, T., Wang, J., Qi, G.-J., and Wang, Z. (2008). Joint multi-label multi-instance learning for image classification. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

Zhu, X. and Bain, M. (2017). B-CNN: branch convolutional neural network for hierarchical classification. *CoRR*, abs/1709.09890.