Semantic Hierarchy-based Convolutional Neural Networks for Image Classification

Matheus Inoue[‡*], Carlos Henrique Forster[*], Antonio Carlos dos Santos[†]
[‡]Polytechnic School, University of São Paulo, São Paulo, Brazil
[*]Computer Science Division, Aeronautics Institute of Technology, São José dos Campos, Brazil
[†]Data Science Team, Itaú-Unibanco, São Paulo, Brazil
minoue@usp.br forster@ita.br antoniocarlos.santos@itau-unibanco.com.br

Abstract—**In this work, three variations of hierarchical topologies of Convolutional Neural Networks (CNNs), two of which being original proposals introduced by this work, were tested to assess their impact on image classification problems. The hierarchical structure groups the images based on the semantic meaning of the classes, from the coarsest classes to the finest classes, forming hierarchical levels. The hierarchical models made were compared to a baseline regular CNN on benchmark image classification datasets, the Fashion-MNIST and CIFAR-100 datasets. Another contribution of this work is a new training strategy for hierarchical CNNs, that aims to be simple to implement and to produce a smooth loss during training, increasing stability, while maintaining characteristics like the transitioning from coarse-to-fine level emphasis during training, learning first high-level details and then specific details that differentiate the fine level classes. The hierarchical models produce outputs for each hierarchical level, which can lead to more interpretable results. Results suggest that providing semantic hierarchies can improve fine level accuracy on CNNs, while bringing relevant hierarchical information from their other coarser level outputs.**

Index Terms—**Convolutional Neural Networks, Hierarchical Image classification, Deep Learning, Computer Vision**

I. Introduction

Convolutional Neural Networks (CNNs) have emerged as a tool to solve Computer Vision problems, in particular, Image Classification, improving the best results obtained by former Image Processing methods [1]. Research shows that CNNs can detect semantic parts on pictures [2], such as wheels on images of cars and trucks, or horns on pictures of moose and cattle, and this semantic detection is used to extract relevant features for image classification.

The use of hierarchical classification on images derives from the idea that humans identify objects based on the context of an image semantic hierarchy, and not by only by its most specific semantic label. For example, certain humans can identify different breeds of cats and correctly label them, like *animal* $\rightarrow$ *cat* $\rightarrow$ *Siamese cat*, while other humans cannot discern between cat breeds, but still make decent predictions, although not as specific, like *animal* $\rightarrow$ *cat*. If it is required to choose a fine class, the breed name, this human can make a mistake for not knowing the correct cat breed, but he will choose a random cat breed, and not a breed that represents other animals, like for example, German shepherd, a dog breed.

However, most image classification CNNs do not consider the possible hierarchical sub-divisions that can be made from the images. For example, the process of distinguishing "dog" from "train" treats them as two totally distinct classes, in the same way as distinguishing "dog" from "wolf", classes that may share some common features. Notice that these CNNs do not include any hierarchical topology and usually use one-hot encoding for labeling, meaning that all labels initially are treated as unrelated to the other classes, and any relationship discovered by the network is due to the data used in the training process. Considering this, one idea to improve the accuracy of image classification CNNs and to extract deeper meaning of the classification is to implement some pre-knowledge to the CNN, such as semantic hierarchies.

A semantic hierarchy categorizes images into fine to coarse classes, resulting in a hierarchy tree. The tree starts at the coarser level nodes, and each child of these nodes represents finer level nodes, until the final level of nodes of the tree, that represents the fine classes of a dataset, for example, Fig. 1 shows an example of a hierarchy tree where each fine class, in red, is grouped on coarse green classes and each coarse class in grouped on two coarser classes in blue. This hierarchy tree forms semantic paths, that demonstrate how a fine class can be grouped with other classes based on proximity of semantic meaning. For Fig. 1, valid semantic paths include:

- *Animal* $\rightarrow$ *Mammals* $\rightarrow$ *Tapir*
- *Man made* $\rightarrow$ *Furniture* $\rightarrow$ *Chair*
- *Animal* $\rightarrow$ *Insects* $\rightarrow$ *Beetle*

Another reasoning behind using hierarchical CNNs is that during training, some weighting for the different hierarchical levels can be set, to give more focus on a certain hierarchical level during some part of the model training. And this weighting process can mimic how humans learn specific topics, for example, an entomologist at the start of his studies probably can identify the different kinds of insects, but can't identify specific species of a given kind of insect, but as he progresses his studies, he will be able to identify the different species of beetles, butterflies etc. Similarly, a hierarchical CNN can start the training trying to classify images between the coarser levels, and as the training progresses, give more focus on the

Any options, findings, and conclusions expressed in this manuscript are those of the authors and do not necessarily reflect the views, official policy or position of the Itaú-Unibanco.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

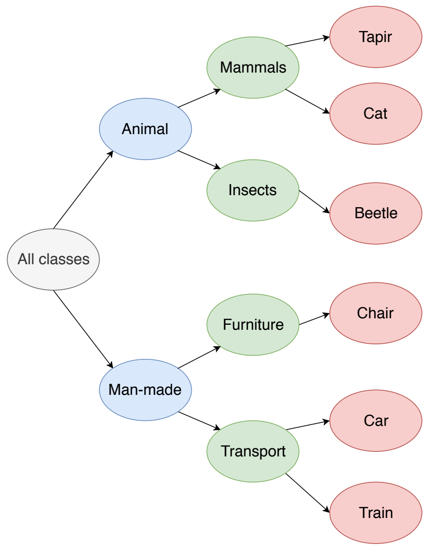


Fig. 1. Example of hierarchical label tree.

fine class classification.

This work contributes to investigate whether the inclusion of a semantic hierarchy of the class labels as input can improve image classification performance. Three different hierarchical topologies were trained from scratch using a semantic hierarchy crafted by the authors of this work on benchmark datasets for image classification, the Fashion-MNIST and CIFAR-100 datasets, and the results were compared to a non-hierarchical baseline CNN. This work also investigates two different methods for training hierarchical CNNs.

In summary, this paper has the following contributions:

- Two new hierarchical topologies for CNNs were proposed (Concat-net and Add-net), to evaluate how hierarchical CNNs trained from scratch perform on image classification tasks when compared to a baseline non hierarchical model.
- A new hierarchical training strategy is proposed, the Adaptive training strategy, that aims to improve current training strategies used in being simple to set and that doesn't lead to sudden spikes on the loss function during training, providing a more stable loss.

II. Related Work

The use of hierarchical structures to improve the results of CNNs for image classification have two main approaches: (1) The works where the hierarchy of classes are predefined by humans, like this work, that can infer semantic level hierarchy to the CNNs, which can be used when the fine level classification is indecisive [3]; (2) and works where the hierarchical trees were constructed based on grouping similar images, using techniques like clustering [4], [5], that can lead to models with improved results over the baseline, but the hierarchy trees may not have any semantic meaning.

One work that represents the state-of-art of hierarchical CNNs is the HD-CNN (Hierarchical Deep CNNs) proposed by Yan et al [6], that is composed by coarse component networks trained over all classes and fine component networks trained on the hierarchical fine classes. One downside of this topology is the general complexity of the network, and the need to pre-train the components of the HD-CNN.

In some literature, Recurrent Neural Networks (RNNs) are employed to implement hierarchical classification [3], [7]. In such works, the recurrent aspect is the hierarchical levels of the images, rather than any time or order based relation, as it is usual when RNNs are used.

Other works use the structure of hierarchical CNNs to produce interpretable models, such as for medical image classification [8], that classifies and detects multiple features from images that lead to a more precise result, or complementary feature detection [9], where the output of feature detection networks are combined to produce a finer detail output.

Other interesting approaches are the bio-inspired works, like [10], that proposed a image classification model based on the two parallel procedures that the brain uses to process visual information, the ventral flow that is responsible for object recognition, and the dorsal flow, that is responsible for spatial information. Both processes were made using CNNs running in parallel.

III. Models

This work tests three different architectures for hierarchical classification, all of them built using regular layers commonly used on CNNs, like convolutional and dense layers. The models are built to have multiple outputs, one for each hierarchical level. The models are trained from scratch, without pre-training on any layer. Fig. 2 shows the basic topology of each model.

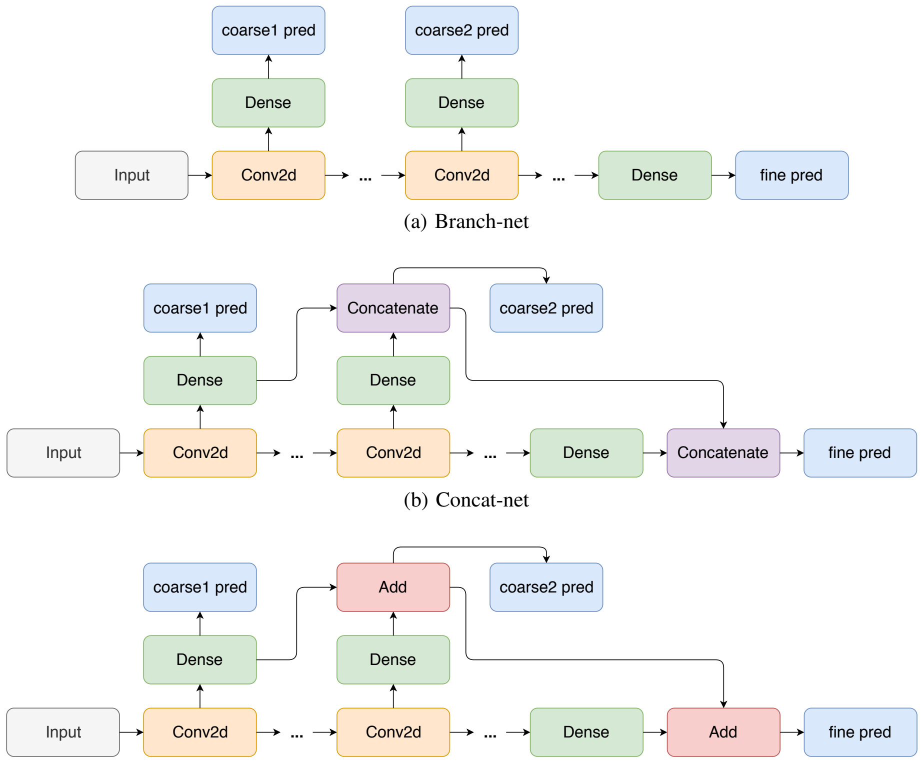
A. Branch-net

The Branch-net, proposed by Zhu and Bain [11] is a hierarchical architecture that can be seen as a regular CNN, that have ramifications on some layers, the coarse branches, that are responsible to classify the high-level hierarchies of the classification problem, as seen on Fig. 2a. Each branch is a sub-network of the main CNN, that classifies the image on its hierarchy level, for example, the first branch will classify the image in the coarser level considered.

The main idea of this topology is to prevent the vanishing gradient problem by having the branches occurring at shallower layers than the main network, and by updating the weights of the shared layers based on the hierarchical features of the images.

B. Concat-net

The Concat-net is a proposed variation of the Branch-net, that have the same overall structure, but for each hierarchical level, the last dense layer will consider the last dense layer of the previous branch, if it exists, concatenating the dense layers, as seen on Fig. 2b. For example, for a three-level Concat-net, the first branch has 16 neurons on the last dense layer, since it is the coarser branch, no modifications are made compared



(a) Branch-net

(b) Concat-net

(c) Add-net

Fig. 2. Hierarchical architectures tested on this work.

to a regular Branch-net, but for the second branch that have 32 neurons on the last dense layer, this layer is concatenated with the last layer of the first branch, meaning that, the last dense layer of this branch have a tensor of $16+32$, 48 values. In the same fashion, for the fine classes, the last hierarchical level, will have its last dense layer concatenated with the last dense layer of the previous level, adding 48 features.

The main idea of this topology is to keep the qualities of the Branch-net and use the concatenations on the branches to directly share relevant features extracted to improve hierarchical classification.

C. Add-net

The Add-net is another variation on the Branch-net, similar to the Concat-net model, but instead of concatenating the last dense layers of each hierarchical levels, it adds the values of each layer as seen on Fig. 2c. For the addition be possible, all of the added layers must have the same length.

The idea behind the addition is that each hierarchical level layer will share information across hierarchical levels based on the position of the added neurons.

IV. Training Strategies

A. Loss Function and Loss Weights

The natural choice for the loss function to classify images between different classes is the Categorical Cross-Entropy loss function (1), since it is desired to work with the maximization of the probability of the correct label for each class, where C is the total number of classes, s_p is the score for the correct label and s_j is the score for the j^{th} label.

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{1}$$

Considering that hierarchical classification deals with N levels of multi-class classifications, one approach is to consider a loss function that is a weighted sum of N functions, for each hierarchy level, weighted by a parameter W_n that is the weight for the n^{th} level (2).

$$Loss = \sum_{n=1}^{N} W_n CE_n \tag{2}$$

The values for $W_1, W_2, \ldots W_N$ should add to 1, going from the coarser level to the finest, so for a three-level hierarchy, a vector of weights $[0.8, 0.2, 0]$ is a valid vector that puts more emphasis for the highest hierarchical level. It is interesting to note that if a model is trained with a vector of weights $[0, 0, \ldots, 1]$ the training will be analogous to train a model considering only the fine class, like a non hierarchical model.

For a model to learn the hierarchy between the classes, two methods were tested to update the vectors of weights during the model training, starting with emphasis on the coarser hierarchies and then focusing on the finest hierarchy as the training progresses, based on a generalization to specialization process to infer hierarchical features from the images, the Stepwise and Adaptive strategies.

Algorithm 1 Callback for Stepwise Strategy with 3 levels of hierarchy

Input: Initial loss weights $(\alpha_0, \beta_0, \gamma_0)$ and list made by tuples composed by the epoch to update values $(ep_1, ep_2, \ldots ep_N)$ and the new loss weights ($\alpha_i, \beta_i, \gamma_i$, for a given epoch ep_i)
Output: Updated loss weights (α, β, γ) for next epoch

```
Initialization:
  α, β, γ ← α_0, β_0, γ_0
  list_epochs ← ep_1, ... ep_N
  list_weights ← (α_1, β_1, γ_1), ...(α_N, β_N, γ_N)
  change_epoch ← ep_1
  i ← 1
on_epoch_end do
  epoch ← get last epoch
  if (epoch == change_epoch) then
    α, β, γ ← list_weights[i]
    i ← i + 1
    change_epoch ← list_epochs[i]
  end if
end on_epoch_end
return α, β, γ
```

B. Stepwise Training Strategy

The Stepwise Training strategy is a method where the vector of weights is updated at fixed given epochs, analogous to the Branch Training strategy method by Zhu and Bain [11], but in this work this strategy was used to train different hierarchical architectures, hence the choice for a different name.

For example, for a two-level hierarchy, the training can start with the vector initiated at $[0.9, 0.1]$, at epoch 25 the vector is updated to $[0.4, 0.6]$ and at epoch 30 the vector is updated to $[0, 1]$, starting with more emphasis in the coarse level, and then changing to emphasize the fine level.

This method is simple to implement, but it has two main issues: A list of all epochs and their weights for each update is needed, which can lead to a large number of hyperparameters to be set, and the update of the weights vector at fixed epochs can lead to sudden spikes of the loss function, that can impact the model training. Algorithm 1 shows an implementation of the Stepwise training strategy, for a three-level model training.

C. Adaptive Training Strategy

The Adaptive Training strategy is an original strategy proposed by this work, that aims to be simple to set and an effective strategy for hierarchical training, focusing in fixing the main issues with the Stepwise strategy by updating the loss weights at every epoch, minimizing possible spikes of the loss and requiring only one hyperparameter, the decay rate τ.

This method is based on the exponential decay of the weights, updating the weights on a pairwise fashion, decreasing the coarser weight until a threshold is reached, while increasing the next hierarchical level weight, and then doing the same thing for the next pair of weights, for example, for a weights vector $[W_1, W_2, W_3]$, the first pair is $[W_1, W_2]$, reducing W_1 while increasing W_2, until W_1 is below a threshold, that will set W_1 to 0 and change the update pair to $[W_2, W_3]$, and the same process continues.

Algorithm 2 Callback for Adaptive Strategy with 3 levels of hierarchy

Input: Initial loss weights $(\alpha_0, \beta_0, \gamma_0)$ and decay rate τ
Output: Updated loss weights (α, β, γ) for next epoch

```
Initialization:
  α, β, γ ← α_0, β_0, γ_0
  decay_rate ← τ
  offset ← 0
  pr ← [1, 2]
on_epoch_end do
  epoch ← get last epoch
  list_w ← [α, β, γ]
  if (epoch > 1) then
    epoch ← get last epoch
    loss_coarse1 ← get current loss for coarse1 classes
    loss_coarse2 ← get current loss for coarse2 classes
    loss_fine ← get current loss for fine classes
    losses ← [loss_coarse1, loss_coarse2, loss_fine]
    ratio ← losses[pr[1]] / (losses[pr[2]] · (epoch + offset))
    decaying ← exp(−ratio · epoch · decay/min(pr)^2)
    increasing ← 1 - decaying
    if (1 − increasing < 0.1) and pr[2] < 3 then
      list_w[pr[1]] ← 0
      list_w[pr[2]] ← 1
      pr ← [pr[1] + 1, pr[2] + 1]
      offset ← offset + epoch
    else
      list_w[pr[1]] ← decaying
      list_w[pr[2]] ← increasing
    end if
    α, β, γ ← list_w[1], list_w[2], list_w[3]
  end if
end on_epoch_end
return α, β, γ
```

The exponential decay uses the information of the current loss for each hierarchical level to calculate the ratio between the loss values of the current update pair ($coarser \div finer$), the decay rate hyperparameter that is set by the user and the value of the current epoch, that acts like an acceleration of the decay rate as the training progresses. Algorithm 2 shows an implementation for a three-level model.

Fig. 3 shows the expected behavior of the progression of the training loss between the two training methods on synthetic data, showing how the Adaptive method can prevent sudden spikes.

V. Experiments

A. Overview

In this work, the experiments compared the hierarchical models with a baseline non hierarchical model on two benchmark image classification datasets, the Fashion-MNIST [12]

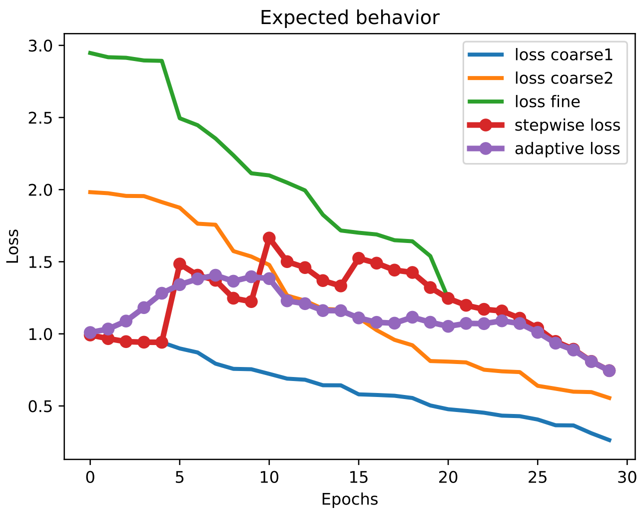


Fig. 3. Expected behavior for the training strategies.

and CIFAR-100 [13] datasets. A baseline architecture was built and all hierarchical models were based on this baseline, so all models have similar sizes. Considering that the Concat-net and Add-net are built like the Branch-net, meaning that only the additions made to built the branches of the Branch-net are reported. Another consideration is that for the Add-net, when required, the number of neurons for each added layers was modified to be the same size of the layer that produces the fine level prediction.

One important observation is that the main objective of this work is to investigate the performance of hierarchical and non-hierarchical CNNs with the same overall structure, and not to build a model to improve the current state of the art on the datasets, more like a viability check for this strategy.

The hierarchical structure made was based on the semantic meaning of the images labels of each dataset tested, and was manually made by the authors of this work.

The datasets were divided into three distinct sets, the training set to train the model, the validation set to fine tune the model during training and the test set that was used only for benchmark purposes.

All the models were trained using Python with the functional Deep Learning API Keras [14], using TensorFlow [15] as the backend, running on the Google Colab[1] cloud infrastructure GPUs, that provide GPU computing free-of-charge, fully configured for deep learning applications [16].

All models were made using regular convolutional and dense layers, using ReLU as the activation function, and for the convolutional layers the filter size 3×3 was used at all times. The models also used max pooling layers to reduce the dimensionality of the images, Dropout [17] layers for regularization and Batch Normalization [18] layers to improve the performance and increase the stability of the network. The models were set to train for 100 epochs, with Early Stopping set to finish the model training when the loss on the validation set don't improve for 15 epochs. The models used Stochastic Gradient Descent (SGD) as the optimization method, with the initial learning rate set to 0.001 and Nesterov momentum of 0.9.

All hierarchical architectures were trained twice for each dataset, one time using the Stepwise training strategy and another using the Adaptive strategy. For the Stepwise strategy, for a given dataset, the update schedule of the weights was the same for all models, while for the Adaptive strategy some models had different values for the decay rate τ, based on how the architecture performed on pre-testing.

B. Fashion-MNIST

The Fashion-MNIST dataset [12] is composed by $70,000$ 28×28 gray-scale images of 10 different classes, divided in $60,000$ training samples and $10,000$ test samples, based on the structure of the MNIST dataset [19]. The training set used was further divided in 80% for training samples and 20% for validation samples.

The 10 classes of the dataset are clothing items, including shirts, sneakers and dresses, that were manually divided in 5 coarse classes, in a two-level hierarchy, as seen on Table I.

TABLE I
HIERARCHY PROPOSED FOR THE FASHION-MNIST DATASET

Coarse classes	Fine classes
Top	T-shirt
	Pullover
	Coat
	Shirt
Bottom	Trouser
Body	Dress
Footwear	Sandal
	Sneaker
	Ankle Boot
Accessory	Bag

The baseline model used two convolutional layers, with 32 and 64 neurons respectively, each one followed by a maxpooling layer, one dense layer with 128 neurons and the output layer with 10 neurons for classification. For the hierarchical models the Branch-net added one branch after the first max pooling layer, with a dense layer of 64 neurons and the output layer for the coarse level classes.

For the hierarchical models trained with the Stepwise training strategy, the loss weights $[\alpha, \beta]$ were initiated at $[0.95, 0.05]$, and updated at epoch 8 to $[0.6, 0.4]$, at epoch 12 to $[0.2, 0.8]$ and at epoch 22 to $[0, 1]$. For the Adaptive method, different values for the decay rate hyperparameter τ were used for each model, 0.3 for Branch-net, 0.4 the Concat-net and 0.4 for the Add-net.

The results obtained are on Table II. The hierarchical models presented similar results on the accuracy of the fine class compared to the baseline, probably due to the shallow hierarchy made for just 10 fine classes, or the overall small network tested.

[1]https://colab.research.google.com/

TABLE II
FASHION-MNIST RESULTS

Model	Method	Test Accuracy(%)	Coarse Accuracy(%)	Δ Test Accuracy(%)
Baseline	-	90.74	-	-
Branch-net	Stepwise	90.81	96.43	0.07
	Adaptive ($\tau = 0.3$)	90.89	**96.75**	0.15
Concat-net	Stepwise	91.14	94.19	0.4
	Adaptive ($\tau = 0.4$)	91.22	95.14	0.48
Add-net	Stepwise	91.57	93.48	0.83
	Adaptive ($\tau = 0.4$)	**91.91**	93.4	**1.17**

C. CIFAR-100

The CIFAR-100 dataset [13] is composed of $60,000$ 32×32 color images divided on 100 different classes, divided in $50,000$ training samples and $10,000$ test samples, totalizing 500 training samples per fine class. For this dataset the training set used was divided in 90% for training samples and 10% for validation samples.

This dataset is interesting for this experiment because it possess a large amount of different classes, that can be grouped on several hierarchical levels, making a good test to check if using hierarchical models can lead to improvement over regular models.

The authors of the dataset grouped the 100 fine classes in 20 coarse categories[2] that were used by this work with two modifications: The proposed coarse classes *vehicles1* and *vehicles2* were merged in one coarse class *vehicles*, and the fine classes *beaver*, *otter* and *seal* were moved from the *Aquatic mammals* coarse class to the coarse class *Medium-sized mammals*. This means the 100 fine classes were grouped in 19 coarse classes, and these 19 coarse classes (Coarse2) were further grouped in 9 super-classes (Coarse1), forming a three-level hierarchy as seen on Table III.

For this dataset the baseline architecture proposed was a deeper network, with 8 convolutional layers and two dense layers, to test the capacity of the hierarchical models on deeper networks. The baseline model starts with two convolutional layers with 64 neurons followed by a max pooling layer, two convolutional layers with 128 neurons and a max pooling layer, two convolutional layers with 256 neurons and a max pooling layer, two convolutional layers with 512 neurons and finally two dense layers with 1024 neurons. The first branch of the Branch-net occurs after the second max pooling layer, with two dense layers with 256 neurons each, and the second branch occurs after the third max pooling layer, with two dense layers with 512 neurons each.

The loss weights $[\alpha, \beta, \gamma]$ were initiated at $[0.9, 0.1, 0]$. The Stepwise strategy updated the weights at epoch 5 to $[0.6, 0.35, 0.05]$, at epoch 10 to $[0, 0.8, 0.2]$, at epoch 15 to $[0, 0.4, 0.6]$ and at epoch 22 to $[0, 0, 1]$. The Adaptive strategy used different values for the hyperparameter τ, 0.05 for the Branch-net, 0.025 for Concat-net and 0.025 for the Add-net.

For this experiment, the hierarchical models obtained better results than the baseline model, as observed on Table IV, with a discernible performance gap compared to the previous experiment, with an improvement of over 5% on most models. Fig. 4 shows an example of the output of the hierarchical models.

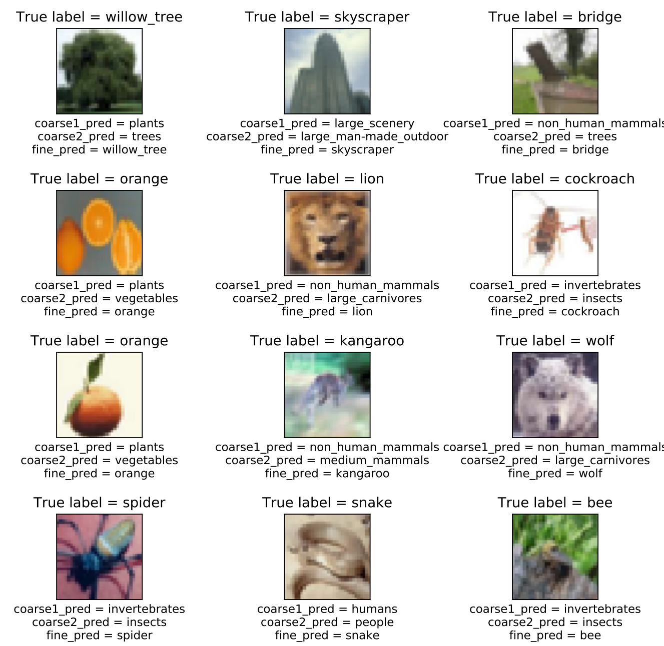


Fig. 4. Sample of the predictions from the Adaptive Add-net model on the CIFAR-100 dataset.

D. Analysis

For the Fashion-MNIST dataset, the hierarchical models had similar performance to the baseline, barely surpassing 1% of accuracy improvement, but the models created with the CIFAR-100 dataset produced more interesting results, with over 5% accuracy improvement, hinting that the hierarchical models work better when deep CNNs are considered, and the dataset have a large number of distinct classes that can be grouped on a semantic hierarchy with more than one level.

Considering the models created for the CIFAR-100, on Fig. 5 it is possible to check how the loss on the validation set progressed during training, and, although slower, all hierarchical models were able to get a smaller loss value than the baseline model on the later epochs of training. This slow start is caused by the hierarchical training strategy, that start focusing on

[2]https://www.cs.toronto.edu/~kriz/cifar.html

TABLE III
HIERARCHY PROPOSED FOR THE CIFAR-100 DATASET

Coarse1 classes	**Coarse2 classes**	**Fine classes**
Aquatic animals	Aquatic mammals	dolphin, whale
	Fish	aquarium fish, flatfish, ray, shark, trout
Invertebrates	Insects	bee, beetle, butterfly, caterpillar, cockroach
	Non insect invertebrates	crab, lobster, snail, spider, worm
Non human mammals	Large carnivores	bear, leopard, lion, tiger, wolf
	Large herbivores	camel, cattle, chimpanzee, elephant, kangaroo
	Medium-sized mammals	fox, porcupine, possum, raccoon, skunk (+ beaver, otter and seal)
	Small mammals	hamster, mouse, rabbit, shrew, squirrel
Humans	People	baby, boy, girl, man, woman
Sauropsida	Reptiles	crocodile, dinosaur, lizard, snake, turtle
Plants	Flowers	orchids, poppies, roses, sunflowers, tulips
	Fruits and vegetables	apples, mushrooms, oranges, pears, sweet peppers
	Trees	maple, oak, palm, pine, willow
Man-made objects	Food containers	bottles, bowls, cans, cups, plates
	Household devices	clock, computer keyboard, lamp, telephone, television
	Household furniture	bed, chair, couch, table, wardrobe
Large objects/scenery	Large man-made things	bridge, castle, house, road, skyscraper
	Large natural scenes	cloud, forest, mountain, plain, sea
Transport	Vehicles	bicycle, bus, motorcycle, pickup truck, train, lawn-mower, rocket, streetcar, tank, tractor

TABLE IV
CIFAR-100 RESULTS

Model	**Method**	**Test Accuracy**(%)	**Coarse1 Accuracy**(%)	**Coarse2 Accuracy**(%)	Δ **Test Accuracy**(%)
Baseline	-	47.02	-	-	-
Branch-net	Stepwise	51.82	62.03	55.39	4.8
	Adaptive ($\tau = 0.05$)	52.3	60.27	57.16	5.28
Concat-net	Stepwise	52.14	62.72	56.3	5.12
	Adaptive ($\tau = 0.025$)	52.32	63.86	**59.9**	5.3
Add-net	Stepwise	51.77	63.25	55.95	4.75
	Adaptive ($\tau = 0.025$)	**52.69**	**64.53**	59.64	**5.67**

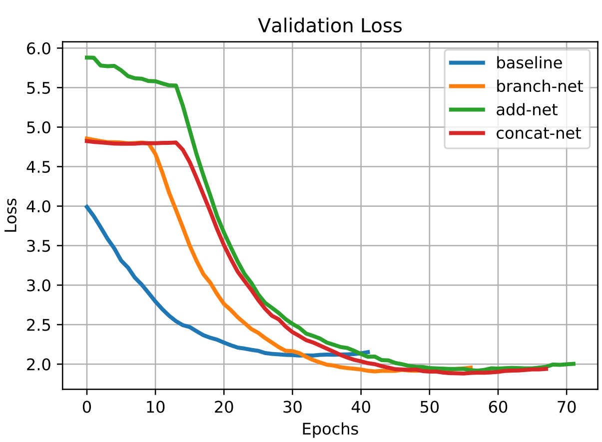


Fig. 5. Validation loss on the validation set for the CIFAR-100 dataset, Adaptive training strategy.

the coarse-level classification, giving a higher value of the loss weight for the coarser level, and progressing to give more focus to the fine level classification. This hints that the hierarchical models training were able to better optimize on the loss function with the coarse-to-fine approach proposed.

The Stepwise and Adaptive training strategies attained similar overall results, with the Adaptive models producing the better models when comparing to its Stepwise model counterpart, on both the fine and coarse level classification. An interesting analysis can be made when we compare how the loss function progresses during training for each strategy, and the influence of how the updates on the loss weights vector are made, presented on Fig. 6 for the Branch-net models, showing a similar result to the expected behavior of the strategies as foreseen on Fig. 3. For each time the Stepwise strategy updated its loss weights, sudden spikes on the loss function occurred, while the Adaptive method, updating the loss weights continuously at every epoch, produced a smoother loss, which leads to a more stable training.

VI. CONCLUSIONS

This work tested three different topologies for hierarchical CNNs, to check if providing semantic hierarchies for the images labels could lead to improvement on the results of image classification problems, and also compared two methods of how to train these hierarchical models.

The results obtained by the experiments suggest that the hierarchical topologies proposed can lead to an improvement of results on image classification tasks. Having a dataset with a large number of classes and that can be hierarchically grouped on several levels, like the CIFAR-100 dataset, seems to be

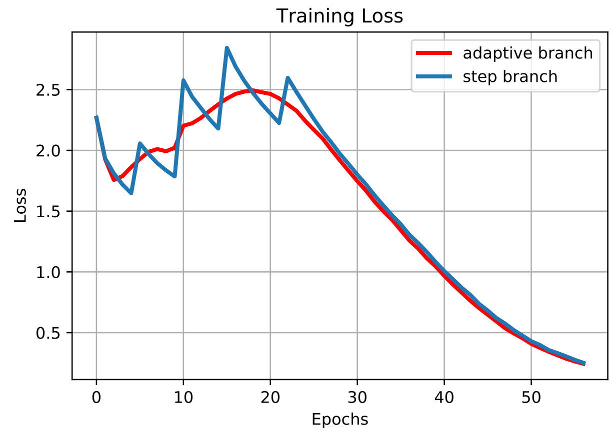


Fig. 6. Comparison of the training loss between training strategies, Branch-net model.

a good indicator of when to try to implement a hierarchical model.

Of all topologies tested, the Add-net, an original topology proposed by this work, obtained the better results overall, performing well across all hierarchical levels. The performance improvement suggests that the hierarchical models ramifications can help to mitigate the vanishing gradient problem of deep CNNs.

The Adaptive training method proposed by this work got the better results when compared to the Stepwise strategy, on the fine and coarse level classification, while being easier to set, requiring only one hyperparameter, showing promising results for future implementations. One observation of the training methods tested, is that since both of them are based on a coarse-to-fine focus, the models require more epochs to reach a similar performance than its non hierarchical counterpart, but the models can get to a smaller loss value.

VII. Further Work

As future work, we plan to implement methods for fine tuning the model, that can work with early stopping methods to build production-ready hierarchical models, like the usage of a method to fine tune the learning rate, like a Cyclical learning rate [20] with regards to hierarchy levels.

Considering production-ready models, another future work is investigate how the hierarchical topologies work when using pre-trained networks, to check the viability of using transfer learning.

The Adaptive training strategy used in this work produced interesting results, but other methods can be tested, like a method that uses more than one decay rate for all hierarchy levels, or a method that updates all loss weights at the same time, not doing in a pairwise fashion as proposed.

From an investigative point of view, it should be interesting to further study some of the details of this work, like the performance of the Stepwise training strategy varying the number of updates of the loss weights, or the influence of misclassification on any of the hierarchical levels.

Acknowledgment

The authors of this paper would like to thank the Aeronautics Institute of Technology (ITA) and Itaú-Unibanco for the Specialization in Data Science course, and for all the support given for the realization of this work.

References

[1] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[2] A. Gonzalez-Garcia, D. Modolo, and V. Ferrari, "Do semantic parts emerge in convolutional neural networks?" *International Journal of Computer Vision*, vol. 126, no. 5, pp. 476–494, 2018.

[3] L. Wang and A. Sohmshetty, "Learning image representations to understand and predict semantic hierarchies," representation Learning in Computer Vision project report.

[4] S. Jiang, T. Xu, J. Guo, and J. Zhang, "Tree-cnn: from generalization to specialization," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, p. 216, 2018.

[5] D. Roy, P. Panda, and K. Roy, "Tree-cnn: A deep convolutional neural network for lifelong learning," *arXiv preprint arXiv:1802.05800*, 2018.

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

[7] Y. Guo, Y. Liu, E. M. Bakker, Y. Guo, and M. S. Lew, "Cnn-rnn: a large-scale hierarchical image classification framework," *Multimedia Tools and Applications*, pp. 1–21, 2018.

[8] S. Shen, S. X. Han, D. R. Aberle, A. A. Bui, and W. Hsu, "An interpretable deep hierarchical semantic convolutional neural network for lung nodule malignancy classification," *arXiv preprint arXiv:1806.00712*, 2018.

[9] S. Hou, X. Liu, and Z. Wang, "Dualnet: Learn complementary features for image recognition," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 502–510.

[10] Y. Yu, K. Hao, and Y. Ding, "A new image classification model based on brain parallel interaction mechanism," *Neurocomputing*, vol. 315, pp. 190–197, 2018.

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

[12] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[14] F. Chollet *et al.*, "Keras," 2015.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[16] T. Carneiro, R. V. M. da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. de Albuquerque, and P. P. Rebouças Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications," *IEEE Access*, 2018.

[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[20] L. N. Smith, "Cyclical learning rates for training neural networks," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 464–472.