Coarse-to-Fine Curriculum Learning for Classification

Otilia Stretcu & Emmanouil Antonios Platanios & Tom Mitchell & Barnabás Póczos
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{ostretcu, e.a.platanios, tom.mitchell, bapoczos}@cs.cmu.edu

Abstract

When faced with learning challenging new tasks, humans often follow sequences of steps that allow them to incrementally build up the necessary skills for performing these new tasks. However, in machine learning, models are most often trained to solve the target tasks directly. Inspired by human learning, we propose a novel curriculum learning approach which decomposes challenging tasks into sequences of easier intermediate goals, that are used to pre-train a model before tackling the original task. We focus on classification tasks, and design the intermediate tasks using an automatically constructed label hierarchy. We train the model at each level of the hierarchy, from *coarse labels to fine labels*, transferring acquired knowledge across these levels. For instance, the model will first learn to distinguish animals from objects, and use this acquired knowledge when learning to classify more fine-grained classes such as "cat", "dog", "car", and "truck". We evaluate our method on several established datasets and show performance gains of *up to 7%* increase in classification accuracy.

1 Introduction

Artificial Intelligence (AI) has seen an impressive leap in the last decade. However, these advances were only possible through big engineering efforts for collecting large amounts of data. Humans, on the other hand, are very good at learning new skills quickly using impressively small amounts of data. Inspired by this, AI researchers have often attempted to create models that resemble the way the human brain works (e.g., LeCun et al., 2015; Vaswani et al., 2017). However, one key difference between human and machine learning that is often overlooked lies not in the model architecture, but in the way that training data is presented to the learner. Unlike AI systems, humans do not simply learn new and difficult tasks (e.g., solving differential equations) from scratch by looking at independent and identically distributed examples of the task being performed by someone else. Instead, new skills are often built progressively, starting with easier tasks and gradually moving to harder ones. For example, students first learn learn to perform addition, multiplication, differentiation, and simple equation solving, before going on to learn about differential equations. Moreover, McClelland & Rogers (2003) showed that human learning of classification problems follows a coarse-to-fine structure, and furthermore that semantic dementia causes cognitive degradations in the reverse order. Thus, we can think of human learning as often being aided by a *curriculum* which may be either provided by a teacher, or learned directly by the student. In this paper, we propose an algorithm for learning and using a curriculum in the context of machine learning.

Using curricula in machine learning was first proposed by Elman (1993). There are multiple lines of work attempting to devise machine learning strategies inspired by human learning in which concepts are being learned in order of increasing difficulty. Most such efforts are focused around scheduling the order in which training data is presented to the learner (e.g., Wang et al., 2018; Zhou & Bilmes, 2018; Jiang et al., 2015; 2018; Bengio et al., 2009). Such strategies may be appropriate for some domains, such as machine translation (Platanios et al., 2019), where we can assume that some training examples are easier than others (e.g., short sentences are easier to translate than long ones), design heuristics for measuring the difficulty of each example. However, we argue that for many common learning tasks, particularly classification tasks, the errors that a model makes are often due to the *similarity of the classes* being considered (e.g., it may be harder to distinguish between a cat and a dog, than between a mammal and a reptile), rather than the absolute difficulty of a sample independent of its class. To exemplify this, we show in Appendix A the confusion matrix of a convolutional neural network (CNN) classifier trained on the popular CIFAR-10 dataset (Krizhevsky et al., 2009). This confusion matrix shows that the mistakes the model makes are not uniformly distributed among all pairs of classes. Instead, they are mostly dominated by a select few class pairs that are difficult to distinguish (e.g., “dog” and “cat”). Moreover, certain classes like “automobile” are mainly confused with only a few other classes, suggesting that a sample is in many cases difficult to classify correctly due to its similarity to a few specific other classes, rather than due to being inherently difficult (e.g., because the input image has a lower signal-to-noise ratio). Therefore, in such cases it may be beneficial to consider the difficulty of the classes—rather than that of the data samples—to more effectively perform curriculum learning.

To this end, we propose a novel algorithm for *performing curriculum learning on the output space of a model*, rather than on its input space. Our algorithm is targeted at classification problems and allows learners to set their own easy goals, towards learning to solve a difficult classification problem. This is motivated by the fact that learners may benefit from learning to classify labels in stages, starting with coarse-grained concepts (e.g., learning to distinguish between “animal” and “object”), before moving on to more fine-grained concepts (e.g., “dog”, “cat”, “car”, “truck”). The proposed algorithm relies on the intuition that the model can *transfer* the knowledge it has about an easy task to better learn to perform a hard task. Our main contribution is a novel algorithm for curriculum learning that can be applied to any classification problem without requiring any extra human supervision, and is thus broadly applicable to many areas of machine learning. Furthermore, it is model-independent, and can thus be used with any model architecture. We perform an empirical evaluation on several established classification datasets and using several types of models, and show that it consistently helps boost performance. The gains are especially prevalent in settings where we have low amounts of training data, resulting in *up to 7%* improvements in classification accuracy.

2 Proposed Method

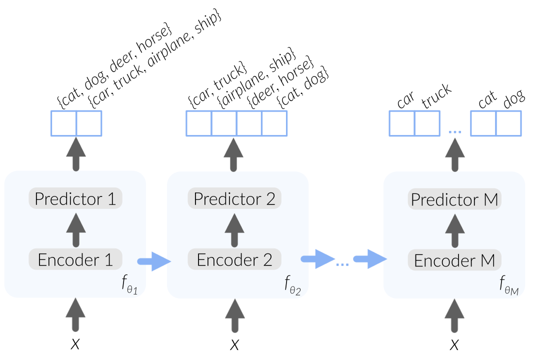

Figure 1: Our proposed learning paradigm. We train multiple functions f_{θ_1}, f_{θ_2}, ..., f_{θ_M} sequentially, initializing the current function parameters with the trained parameters of the previous one.

Consider a classification task that involves K mutually-exclusive classes. Given a dataset of supervised examples, $\{x_i, y_i\}_{i=1}^N$, our goal is to learn a classification function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ that is parameterized by θ. $\mathcal{X}$ can refer to an arbitrary domain of inputs (e.g., images, text sentences) and $\mathcal{Y} = \mathbb{1}^K$ contains the one-hot encoded representation of the target class for each sample. In what follows, we denote the set of training examples $\{x_i, y_i\}_{i=1}^N$ as two tensors, $\mathbf{X}$ and $\mathbf{Y}$, that contain all of the training examples stacked along the first dimension. The standard strategy for learning f_θ is to initialize θ with random values and iteratively update it by performing gradient descent on a loss function that is defined over $\mathbf{X}$ and $\mathbf{Y}$. In this work, we propose a different approach: we learn a series of auxiliary functions f_{θ_1}, f_{θ_2}, ..., f_{θ_M}, sequentially and one at a time, where the final function f_{θ_M} will become our target function, f_θ. The auxiliary functions f_{θ_1},..., $f_{\theta_{M-1}}$ operate on the same input domain as f_θ, but the classification task that they are performing is *coarser*, meaning that they each learn to classify samples into fewer classes than the one that comes after them. This means that f_{θ_1} is learning an *easier* task than f_{θ_2}, which is learning an easier task than f_{θ_3}, etc., up until f_{θ_M} which is learning our actual target task. Our method thus consists of two parts: (i) deciding on what the auxiliary tasks should be and providing a way to generate training data for them, and (ii) providing a way for each learned function to transfer its acquired knowledge to the function in the chain.

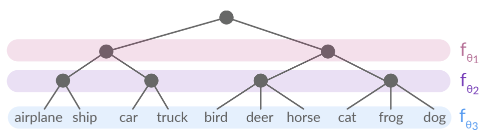

Figure 2: Example label hierarchy generated for the CIFAR-10 dataset. The colored areas indicate the labels that are considered when training each of the functions f_{θ_1}, f_{θ_2}, etc.

Generating Auxiliary Tasks. Our main requirement for the auxiliary tasks is that they form a sequence of increasing difficulty. Luckily there exists a natural heuristic for gauging how difficult a task is, and that is to look at the *confusion matrix* of a trained model. We define this confusion matrix as $\mathbf{C} \in [0, 1]^{K \times K}$ where $\sum_{j=1}^K \mathbf{C}_{ij} = 1$ and $\mathbf{C}_{ij}$ is the probability that the model predicts class j when it should have predicted class i. Given an existing model, this matrix can be approximated using the sample estimate of each probability on a validation dataset. Given the original set of classes and the confusion matrix of a pre-trained model, we expect that: (i) grouping classes that are often confused together to form clusters of classes, and (ii) defining a new classification task where the goal is to predict the cluster instead of the specific class, should result in an easier task. In fact, we can create a sequence of such functions f_{θ_1}, f_{θ_2}, ..., f_{θ_M} of increasing difficulty in the following way. First, we can use $1 - \mathbf{C}$ as input to a hierarchical clustering algorithm to form a label hierarchy, where the labels that are confused more often are connected lower in the hierarchy. An example of such a hierarchy is shown in Figure 2. In our experiments, we used a recent hierarchical clustering algorithm proposed by Bateni et al. (2017) and called *affinity clustering*. We chose this because it ensures that the depth of the hierarchy is at most $\mathcal{O}(\log K)$, where K is the original number of classes, which is important in our case, as will become clear later. Then, then we can consider each level of this tree as an auxiliary task, and form a sequence of tasks by obtaining these levels in top-to-bottom order. In Figure 2, the first auxiliary task f_{θ_1} solves a binary classification problem with 2 classes consisting of the clusters {"airplane", "ship", "car", "truck"} and {"bird", "deer", "horse", "cat","frog","dog"}, and then the following task f_{θ_2} further splits these clusters resulting in 4 classes. These tasks will be trained in order, starting with the top level in the tree (i.e., $l = 1$), and transferring acquired knowledge from each level to the next. A high-level overview of the procedure is illustrated in Figure 1, and our complete algorithm for generating class hierarchies is shown in Algorithm 1 in Appendix B.

Transferring Acquired Knowledge. To transfer knowledge from a trained classifier f_{θ_l} to $f_{\theta_{l+1}}$, for $l \in \{1, \ldots, M-1\}$, we initialize the parameters of $f_{\theta_{l+1}}$ based on the parameters of the trained f_{θ_l} model. Let us assume that: $f_\theta = \underbrace{f^H_{\theta^H}}_{\texttt{predictor}} \circ \underbrace{f^{H-1}_{\theta^{H-1}} \circ \cdots \circ f^1_{\theta^1}}_{\texttt{encoder}}$, where $\circ$ denotes function composition, H is the number of layers of the network, and $\theta = \{\theta^1, \ldots, \theta^H\}$. This is a simple decomposition that applies to most deep learning models used in practice. Intuitively, the encoder converts the input features to a high-level latent representation that is then processed by the predictor to produce a probability distribution over classes. We denote the parameters of the predictor by $\theta^{\texttt{predictor}}$ and those of the encoder by $\theta^{\texttt{encoder}}$. In our experiments, the predictor is simply the output layer of a neural network, whose output dimensionality changes at every hierarchy level, depending on the number of clusters for that level. When training $f_{\theta_{l+1}}$, we initialize its encoder as $\theta^{\texttt{encoder}}_{l+1} = \theta^{\texttt{encoder}}_{l}$. The predictor parameters $\theta^{\texttt{predictor}}_{l+1}$ are initialized randomly. Thus, knowledge transfer in our regime happens through the initialization of the encoder, which often includes most of the model parameters. Figure 1 illustrates this approach. θ_1 is initialized randomly.

Putting the Pieces Together. Previously, we described the three main components of the proposed approach: (i) constructing a class hierarchy, (ii) automatically generating auxiliary tasks and training data for each one of them, and (iii) transferring acquired knowledge between learned functions for the different auxiliary tasks. In Algorithm 3 from Appendix B, we show how these three components fit together and provide an overview of the proposed coarse-to-fine curriculum learning algorithm.

3 Experiments

We performed experiments on both synthetic and real datasets. For all experiments, we used a convolutional neural network (CNN) with 3 convolution layers followed by a single densely connected layer. We first trained the baseline model without using any curriculum and computed its confusion matrix, which is then used to generate a label hierarchy. Next, we trained the model at each level of the hierarchy, top-down, using the nodes at the current level as class labels. Details on the CNN architecture, loss function, optimizer, batch size and other training parameters can be found in Appendix C.

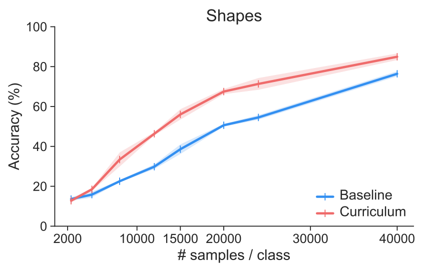

Figure 3: Accuracy mean and standard error for the baseline and the curriculum model, averaged over 5 runs.

Synthetic Datasets. In order to study different properties of the proposed method, we created a synthetic dataset where a natural coarse-to-fine curriculum might arise and that is easy to analyze. We refer to this dataset as Shapes. The inputs consist of 64×64 images depicting geometrical figures. Each image contains one of 10 distinct shapes (circles, ellipses and regular polygons with 3-10 vertices) of one of three colors (magenta, cyan blue or grey), and is placed against a black background. An example is shown in Appendix D, as well as further details on the dataset generation. We have made both the generated dataset and our dataset generation code available at `https://github.com/otiliastr/coarse-to-fine-curriculum`. The results, shown in Figure 3,

Table 1: Results on real classification datasets. We show the accuracy mean and standard error for the baseline and the model trained with curriculum, over 5 runs initialized with random seeds, as well as their difference, computed per run and then averaged.

Dataset	#Classes	#Samples	Accuracy		
			Baseline	Curriculum	Difference
CIFAR-10	10	50,000	71.08 $\pm$ 0.22	**71.81 $\pm$ 0.07**	0.73 $\pm$ 0.25
CIFAR-10	10	20,000	**64.99 $\pm$ 0.25**	64.92 $\pm$ 0.34	-0.07 $\pm$ 0.41
CIFAR-10	10	10,000	60.94 $\pm$ 0.19	**61.42 $\pm$ 0.27**	0.48 $\pm$ 0.19
CIFAR-10	10	5,000	55.89 $\pm$ 0.37	**56.84 $\pm$ 0.42**	0.95 $\pm$ 0.36
CIFAR-10	10	1,000	42.52 $\pm$ 0.64	**43.78 $\pm$ 0.23**	1.26 $\pm$ 0.61
CIFAR-10	10	500	37.75 $\pm$ 0.56	**39.23 $\pm$ 0.68**	1.47 $\pm$ 0.37
CIFAR-10	10	100	24.81 $\pm$ 0.76	**27.69 $\pm$ 0.31**	2.87 $\pm$ 0.80
CIFAR-100	100	50,000	36.56 $\pm$ 0.35	**40.93 $\pm$ 0.25**	4.38 $\pm$ 0.25
CIFAR-100	100	20,000	29.68 $\pm$ 0.18	**32.82 $\pm$ 0.63**	3.14 $\pm$ 0.55
CIFAR-100	100	10,000	24.68 $\pm$ 0.79	**27.66 $\pm$ 0.26**	2.99 $\pm$ 0.73
CIFAR-100	100	5,000	18.26 $\pm$ 0.36	**22.50 $\pm$ 0.48**	4.24 $\pm$ 0.41
CIFAR-100	100	1,000	9.45 $\pm$ 0.27	**11.01 $\pm$ 0.37**	1.56 $\pm$ 0.22
Tiny-ImageNet	200	100,000	18.95 $\pm$ 0.87	**25.43 $\pm$ 0.20**	6.48 $\pm$ 0.67
Tiny-ImageNet	200	50,000	15.26 $\pm$ 0.25	**22.20 $\pm$ 0.42**	6.94 $\pm$ 0.18
Tiny-ImageNet	200	20,000	12.14 $\pm$ 0.15	**15.65 $\pm$ 0.05**	3.51 $\pm$ 0.17
Tiny-ImageNet	200	10,000	8.51 $\pm$ 0.06	**10.85 $\pm$ 0.29**	2.35 $\pm$ 0.35
Tiny-ImageNet	200	5,000	6.34 $\pm$ 0.11	**7.94 $\pm$ 0.15**	1.60 $\pm$ 0.13

indicate that our approach outperforms the baseline consistently, except for when the baseline is very poor (13% accuracy). After investigating into what caused this we made two observations: (i) when the baseline performs too poorly, the confusion matrix used to obtain our class hierarchy is also very poor, resulting in the curriculum method not offering a significant performance boost, and (ii) there is a point at which the amount of labeled data is too little to actually allow any model to learn something meaningful; at that point we do not expect our curriculum approach to offer any significant boost in performance over the baseline method. Figure 3 also shows that our curriculum approach provides the biggest boost over the baseline method in the middle regime, in which there are not enough samples for the baseline method to reach high accuracy, but there is enough to make it a sufficiently good learner that the curriculum learning algorithm can improve upon. These results also agree with previous results showing that pre-training is most beneficial in problems where labeled data is scarce. Moreover, we also inspected the generated label hierarchy. The most common hierarchy obtained during the experiments is shown in Appendix E. Interestingly, this hierarchy is very intuitive and compatible with what a human might have manually constructed; the first level separates all shapes by color, and the second level further groups the shapes based on similarity (circles and ellipses are grouped together, and all polygons are grouped together).

Real Datasets. We perform experiments on the CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky, 2009), SVHN (Netzer et al., 2011) and Tiny-ImageNet (Li et al.) datasets. CIFAR-100 contains labels at two levels of granularity, 100 fine-grained labels and 20 coarse-grained labels, and we experiment with both. Dataset statistics are shown in Appendix G. We use the same model and setup as before. The results for CIFAR-10, CIFAR-100 and Tiny-ImageNet are shown in Table 4, and results for the remaining datasets are shown in Appendix H. Our approach is able to boost baseline performance for all datasets, especially in the cases with limited training data. On the datasets with a larger number of classes, CIFAR-100 and TinyImageNet, our approach consistently outperforms the baseline by a significant margin, regardless of the amount of training data. To understand how the curriculum has been trained, we show one of the most commonly generated label hierarchies for CIFAR-10 in Figure 2, and for CIFAR-100 in Appendix I. Even for CIFAR-100 where we have 100 labels, the hierarchy only contains two auxiliary levels with 6 and 27 clusters, respectively, meaning that we would have to pay at most 3 times the computation time for these test accuracy improvements. In practice, the actual computation time is much less than that, because each hierarchy level now needs fewer iterations to converge than the baseline, as shown in Figure 4. Nevertheless, this extra computation occurs only during training, and there is no extra cost at inference time.

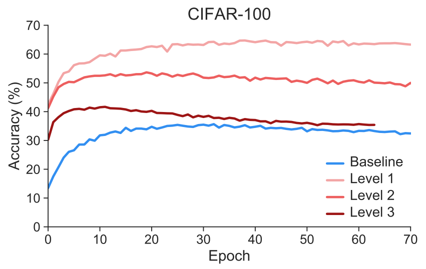

Figure 4: Accuracy per epoch for each of the curriculum hierarchy levels, on the CIFAR-100 dataset.

Acknowledgements

This material is based upon work supported by AFOSR FA95501710218, NSF IIS1563887, DARPA/AFRL and FA87501720130. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Office of Scientific Research, the National Science Foundation, the Defense Advanced Research Projects Agency or Air Force Research Laboratory.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Mohammadhossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems*, pp. 6864–6874. 2017.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *International Conference on Machine Learning*, pp. 41–48. ACM, 2009.

Jeffrey L Elman. Learning and Development in Neural Networks: The Importance of Starting Small. *Cognition*, 48(1):71–99, 1993.

Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-Paced Curriculum Learning. In *AAAI Conference on Artificial Intelligence*, 2015.

Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. 2018.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*. ACM, 2015.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, Massachusetts Institute of Technology and New York University, 2009.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL `https://doi.org/10.1038/nature14539`.

Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Tiny ImageNet Visual Recognition Challenge. `https://tiny-imagenet.herokuapp.com/`.

James L McClelland and Timothy T Rogers. The Parallel Distributed Processing Approach to Semantic Cognition. *Nature Reviews Neuroscience*, 4(4):310–322, 2003.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. 2011.

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. Competence-based Curriculum Learning for Neural Machine Translation. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Cheng Wang, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. Mancs: A Multi-Task Attentional Network with Curriculum Sampling for Person Re-Identification. In *European Conference on Computer Vision*, pp. 365–381, 2018.

Tianyi Zhou and Jeff A Bilmes. Minimax Curriculum Learning: Machine Teaching with Desirable Difficulties and Scheduled Diversity. In *International Conference on Learning Representations*, 2018.

APPENDIX

A CONFUSION MATRIX ON THE CIFAR-10 DATASET

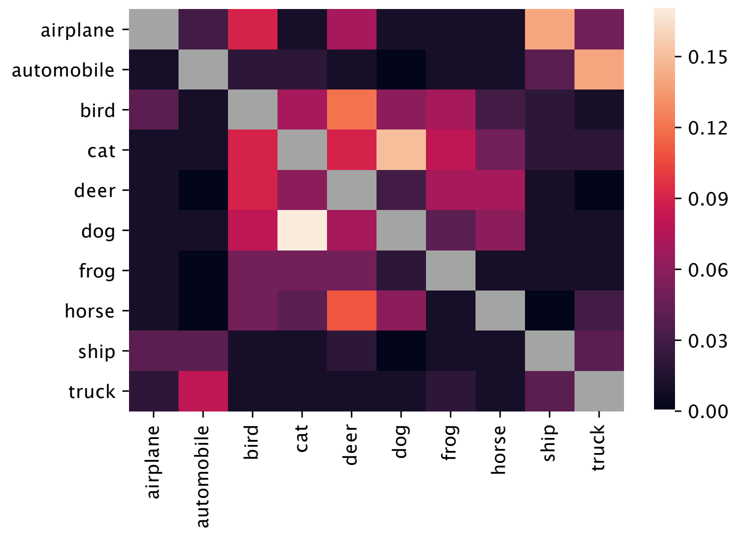

Figure 5: Confusion matrix for a CNN classifier on the CIFAR-10 dataset. Each element at position (i, j) indicates the ratio of times the model wrongly classifies an image as class j instead of the correct class i. The diagonal elements have been removed for visualization purposes.

B ALGORITHMS

Algorithm 1: Generate Class Hierarchy

```
// This algorithm generates a class hierarchy.
Inputs: Number of classes $K$.
        Training data $\{x_i, y_i\}_{i=1}^N$.
        Trainable baseline model $f_\theta$.
1  Train $f_\theta$ on the provided training data $\{x_i, y_i\}_{i=1}^N$.
2  Make predictions $\hat{\mathbf{Y}}$ on a validation set using $f_\theta$.
3  Estimate confusion matrix $\mathbf{C} \in [0,1]^{K \times K}$ using $\hat{\mathbf{Y}}$.
4  Compute the cluster hierarchy, $\mathcal{H}$, by applying affinity clustering on the
     original class, using $1 - \mathbf{C}$ as the matrix of pairwise distances between
     them.
5  clustersPerLevel ← []
6  for $l \leftarrow 1, \ldots,$ depth($\mathcal{H}$) do
7      clustersPerLevel[$l$] ← []
8      foreach $n \in \mathcal{H}$.nodesAtDepth[$l$] do
9          Create cluster $c$ by grouping the leaves of the sub-tree rooted at $n$.
10         clustersPerLevel[$l$].append($c$)
Output: clustersPerLevel.
```

Algorithm 2: Transform Labels

```
// This algorithm computes the transformed target labels for the
// provided clusters.
Inputs: Original labels $\{y_i\}_{i=1}^N$.
        Set of clusters $\{c_{\hat{k}}\}_{\hat{k}=1}^{K}$, where
        each cluster $c_{\hat{k}}$ is a set of labels.
1  originalToNew ← Zero-initialized array of length K.
2  for $\hat{k} \leftarrow 1, \ldots, \hat{K}$ do
3      foreach Label $l \in c_{\hat{k}}$ do
4          originalToNew[$l$] ← $\hat{k}$
5  newLabels ← Zero-initialized array of length $N$.
6  for $i \leftarrow 1, \ldots, N$ do
7      newLabels[$i$] ← originalToNew[$y_i$]
Output: newLabels.
```

Algorithm 3: Coarse-To-Fine Curriculum

```
// This is an overview of the proposed algorithm.
Inputs: Number of classes $K$.
        Training data $\{x_i, y_i\}_{i=1}^N$.
        Trainable baseline model $f_\theta$.
1  clustersPerLevel ← GenerateClassHierarchy( $K, \{x_i, y_i\}_{i=1}^N, f_\theta$)
2  M ← clustersPerLevel.length
   // Train the model at each level of the hierarchy.
3  originalLabels ← [1,...,K]
4  for $l \leftarrow 0, \ldots,$ M - 1 do
5      clusters ← clustersPerLevel[$l + 1$]
6      newLabels ← TransformLabels( $\{y_i\}_{i=1}^N$, clusters)
7      if $l = 0$ then
8          $\theta_{l+1}^{\texttt{encoder}}$ ← random().
9      else
10         $\theta_{l+1}^{\texttt{encoder}} \leftarrow \theta_{l}^{\texttt{encoder}}$
11     $\theta_{l+1}^{\texttt{predictor}}$ ← random().
12     Train $f_{\theta_{l+1}}$ using newLabels as the target labels.
Output: $f_{\theta_{[\texttt{M}]}}$.
```

C Experimental Details

The Convolutional Neural Network used in our experiments contains the following layers:

1. Convolution: 2D convolution using a 3×3 filter with 32 channels, followed by ReLU activation.
2. Max pooling using a 2×2 window.
3. Convolution: 2D convolution using a 3×3 filter with 64 channels, followed by ReLU activation.
4. Max pooling using a 2×2 window.
5. Convolution: 2D convolution using a 3×3 filter with 64 channels, followed by ReLU activation.
6. Fully connected layer performing a linear projection to the output dimension (i.e. number of classes), returning logits.

We implemented our method on top of the TensorFlow framework (Abadi et al., 2015). All models were trained by minimizing the softmax cross-entropy loss function using the Adam optimizer (Kingma & Ba, 2015). We use the default TensorFlow parameters for the optimize rand set the learning rate to 0.001 for a batch size of 512 samples. We also employed early stopping by terminating training when validation accuracy did not improve within the last 50 epochs. We report test accuracy statistics for the iteration that corresponds to the best validation set performance. The validation dataset is obtained by setting aside 20% of the training examples, chosen randomly. Finally, note that all our experiments were performed using a single Nvidia Titan X GPU, and code for reproducing our results is available at `https://github.com/otiliastr/coarse-to-fine-curriculum`.

D Shapes Dataset

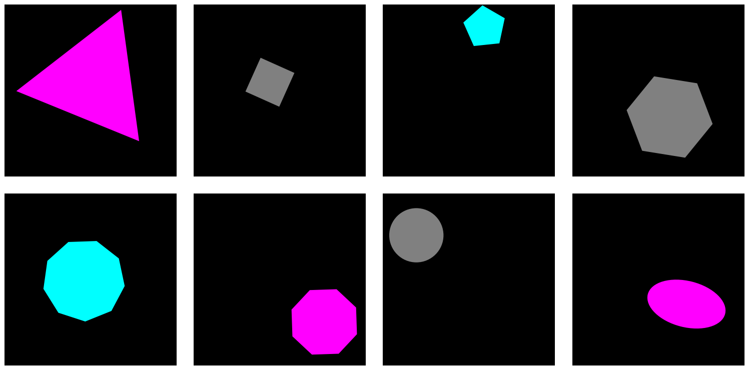

Figure 6: Example of geometrical shapes included in the dataset.

The position of each shape within an image, its size and its rotation angle are chosen randomly from a pre-specified range, as well as its color. The generated dataset contains $50,000$ samples ($5,000$ per shape), split into $40,000$ training and $10,000$ testing samples. Given a dataset that contains a fixed number of samples per shape, the task is to classify the specific geometrical figure present in each image. Shapes of different color have distinct labels (i.e., the model must learn to distinguish between a grey and magenta triangle). Thus, this classification task involves 30 classes.

The idea behind this generation process is that polygons with similar color and number of vertices should look more alike and thus be more prone to be confused by the model (and similarly for circles and ellipses). This kind of confusion forms the main motivation for the proposed method and thus we expect our method to provide a significant performance boost in this setting.

E Generated Label Hierarchy for Shapes

The most common hierarchy generated during the experiments is the following:

Level 1: The shapes are grouped by color into 3 clusters.

Level 2: There are 6 clusters:

- [cyan circle,cyan ellipse],
- [cyan polygon^3,cyan polygon^4,...,cyan polygon^{10}],
- [gray circle,gray ellipse],
- [gray polygon^3,gray polygon^4,...,gray polygon^{10}],
- [magenta circle,magenta ellipse],
- [magenta polygon^3,magenta polygon^4,...,magenta polygon^{10}],

where polygon^k denotes a polygon with k vertices.

Level 3: Each shape has its own class.

F Distance Metric Evaluation

A natural question to ask is whether using the confusion matrix as a distance metric between classes is better than alternative approaches. For example, what if we force classes that are easily confused into different clusters early on? Could that help the model focus on them earlier and learn to handle them better? To answer these questions, we also tested our approach by replacing the confusion matrix with a similarity matrix (i.e., 1 - confusion matrix), and also by using a completely random grouping of the classes. The results are denoted as "Similarity" and "Random", respectively, and are presented in Table 2. Interestingly, the curriculum obtained using either of these approaches harms the baseline performance. On the other hand, our confusion matrix formulation results in a large performance boost. This suggests that using arbitrary class hierarchies and training using our curriculum method is not sufficient; *the actual choice of class hierarchy matters*.

Table 2: Results on the Shapes dataset with $40,000$ samples, comparing different types of distance metrics used to generate the curriculum hierarchy. We show the accuracy mean and standard error of the baseline model, that of the model trained with curriculum, and the average difference between the two (calculated separately per run and then averaged).

Distance	Accuracy		
	Baseline	Curriculum	Difference
Confusion	66.69 ± 1.51	79.52 ± 3.22	11.99 ± 3.43
Similarity	66.69 ± 1.51	60.56 ± 2.77	-6.13 ± 4.01
Random	66.69 ± 1.51	60.77 ± 4.75	-6.19 ± 5.62

G Dataset Information

Table 3: Statistics for the multi-class classification datasets used in our experiments.

Dataset	# classes	# train	# test
Shapes	30	40,000	10,000
CIFAR-10	10	50,000	10,000
CIFAR-100 Coarse	20	50,000	10,000
CIFAR-100	100	50,000	10,000
SVHN	10	73,257	26,032
Tiny-ImageNet	200	100,000	10,000

Table 3 shows the number classes, number of training instances, and number of test instances for all the datasets used in our experiments. The data and specific train/test splits for CIFAR-10, CIFAR-100 and SVHN were obtained from the `tensorflow_datasets` package[1] from TensorFlow. For Tiny-ImageNet, we used the provided train/validation/test splits from Li et al..

H Further Results

Table 4: Results on real classification datasets. We show the accuracy mean and standard error for the baseline and the model trained with curriculum, over 5 runs initialized with random seeds, as well as their difference, computed per run and then averaged.

Dataset	#Classes	#Samples	Accuracy		
			Baseline	Curriculum	Difference
SVHN	10	73,257	**89.79 ± 0.12**	89.63 ± 0.10	-0.17 ± 0.04
SVHN	10	20,000	**86.06 ± 0.28**	85.88 ± 0.39	-0.17 ± 0.63
SVHN	10	10,000	**84.03 ± 0.32**	83.74 ± 0.39	-0.19 ± 0.09
SVHN	10	5,000	**81.66 ± 0.13**	81.51 ± 0.54	-0.15 ± 0.42
SVHN	10	1,000	69.21 ± 0.27	**72.46 ± 0.31**	3.25 ± 0.44
SVHN	10	500	60.06 ± 0.82	**64.71 ± 0.76**	4.65 ± 0.25
SVHN	10	100	20.27 ± 0.90	**20.79 ± 1.30**	2.07 ± 0.84
CIFAR-100 Coarse	20	50,000	49.55 ± 0.32	**50.38 ± 0.13**	0.69 ± 0.35
CIFAR-100 Coarse	20	20,000	**42.27 ± 0.17**	42.04 ± 0.15	-0.23 ± 0.31
CIFAR-100 Coarse	20	10,000	38.96 ± 0.12	**39.12 ± 0.28**	1.15 ± 0.38
CIFAR-100 Coarse	20	5,000	33.52 ± 0.39	**35.12 ± 0.25**	1.60 ± 0.50
CIFAR-100 Coarse	20	1,000	22.21 ± 0.60	**22.79 ± 0.24**	0.59 ± 0.36
CIFAR-100 Coarse	20	500	18.54 ± 0.21	**19.55 ± 0.56**	1.01 ± 0.74
CIFAR-100 Coarse	20	100	11.28 ± 0.49	**12.49 ± 0.40**	1.21 ± 0.11

[1] `https://www.tensorflow.org/datasets/catalog/`

I Generated Label Hierarchy for CIFAR-100

- Level 1:

Cluster 1 :	apple, pear, sweet_pepper, orange, aquarium_fish, sunflower, rose, orchid, tulip, poppy, crab, lobster
Cluster 2 :	baby, woman, girl, hamster, boy, man, fox, lion, snail, camel
Cluster 3 :	flatfish, ray, shark, turtle, dolphin, whale, bear, chimpanzee, skunk, cattle, dinosaur, elephant, seal, otter
Cluster 4 :	crocodile, lizard, shrew, beaver, porcupine, mushroom, kangaroo, tiger, leopard, trout, possum, wolf, mouse, raccoon, squirrel, rabbit
Cluster 5 :	lamp, cup, worm, chair, bed, table, keyboard, couch, snake, bicycle, motorcycle, can, telephone, television, bottle, wardrobe, bowl, plate, clock
Cluster 6 :	caterpillar, bee, butterfly, cockroach, spider, beetle
Cluster 7 :	willow_tree, forest, oak_tree, palm_tree, pine_tree, maple_tree, skyscraper, rocket, tractor, train, tank, castle, bridge, house, streetcar, pickup_truck, bus, lawn_mower, mountain, cloud, road, sea, plain

- Level 2:

Cluster 1 :	apple, pear, sweet_pepper, orange
Cluster 2 :	aquarium_fish, sunflower, rose, orchid, tulip, poppy
Cluster 3 :	bowl, plate, clock
Cluster 4 :	castle, bridge, house
Cluster 5 :	streetcar, pickup_truck, bus, lawn_mower
Cluster 6 :	fox, lion, snail, camel
Cluster 7 :	skunk, cattle, dinosaur, elephant
Cluster 8 :	mountain, cloud, road, sea, plain
Cluster 9 :	crab, lobster
Cluster 10 :	crocodile, lizard
Cluster 11 :	lamp, cup
Cluster 12 :	flatfish, ray, shark, turtle, dolphin, whale
Cluster 13 :	baby, woman, girl, hamster, boy, man
Cluster 14 :	willow_tree, forest, oak_tree, palm_tree, pine_tree, maple_tree
Cluster 15 :	mushroom, kangaroo, tiger, leopard, trout
Cluster 16 :	possum, wolf, mouse, raccoon
Cluster 17 :	seal, otter
Cluster 18 :	squirrel, rabbit
Cluster 19 :	skyscraper, rocket
Cluster 20 :	tractor, train, tank
Cluster 21 :	bear, chimpanzee
Cluster 22 :	shrew, beaver, porcupine
Cluster 23 :	worm, chair, bed, table, keyboard, couch, snake
Cluster 24 :	caterpillar, bee, butterfly
Cluster 25 :	cockroach, spider, beetle
Cluster 26 :	bicycle, motorcycle
Cluster 27 :	can, telephone, television, bottle, wardrobe

- Level 3:
  Each class has its own cluster.