Distilling Knowledge to Specialist Networks for Clustered Classification

Nathanael Romano
Stanford University
naromano@stanford.edu

Robin Schucker
Stanford University
schucker@stanford.edu

Abstract

Most realistic datasets for computer vision tasks tend to have a large number of classes, which are unevenly distributed in the label space, and can even be clustered in categories, like in popular benchmark datasets such as ImageNet or CIFAR-100. Typical convolutional neural networks often fail to generalize well on these datasets, especially when the number or image per class is small. A natural idea, when one does not want to work with huge networks that are impossible to transfer to small devices (both for memory and time constraints), would be to train an ensemble of experts, each one specialized on a subset of the dataset's classes. However, those expert networks tend to overfit a lot. To address this issue, we propose to leverage the concept of knowledge distillation, recently proposed by Hinton et al. *[2], to train those networks. This technique can act as a very strong regularizer, and can allow us to achieve good results on this type of dataset, with a significant speed-up (both for training and prediction) and memory gain.*

After introducing the theoretical foundations of knowledge distillation, we present the different components of the necessary pipeline in the case of specialist networks, and various ways of improving the results. We also show and discuss our experiments and results on a particular dataset, CIFAR-100, which classes presents a natural clustered structure.

1. Introduction

Recent advances in computer vision have allowed Convolutional Neural Networks to achieve extremely high performances on object recognition tasks, almost beating humans on popular benchmark datasets. However, in order to keep high performance when the number of classes to distinguish becomes larger, the networks have to become extremely large and deep, making forward computation slow and nearly impossible on portable embedded devices. When computational power is limited, we are restrained to smaller network architectures but training them from scratch in a traditional way does not utilize the full potential of recent computer vision techniques that enabled those high performances. Indeed, by achieving such high performances in classification, the big cumbersome models (later on denoted as *masters*) have learned much about the internal structure of image data. The purpose of this study is to find the best way to extract that knowledge in order to train smaller models (later on called *specialists*) that specialize in classifying only subsets of the master's classes.

While image classification challenges require a neural network to be able to classify correctly a large number of classes, the best performance is always achieved by a very large network that recognises all the classes rather than a multitude of smaller networks that specialize in a subsets of of the classes. For example, a network that classifies only animals and another that classifies only cars performs always worse that a network that is twice the size but classifies both cars and animals. Indeed, the bigger networks gains on classifying car by seeing the animal data. This pushed the current trend, that *bigger* with *more data* is *better*.

However in the context of limited computation and a goal that is not necessarily recognizing all of the classes (for example only classifying cars), how can we take advantage of the large amount of labeled data available and the knowledge learned of powerful master ensembles? The answer is knowledge distillation. Before making a prediction about the class of an image, the master network produces a set of raw scores (later on called *logits*), which combined with a Softmax, creates a probability distribution over all the classes (cf. Figure 1). While we only care about the maximum probability to produce a prediction, there is a lot of meaningful information in this probability distribution as it can, for example, quantify how much a cat image resembles a dog image rather than a pickup truck image.

Using this approach, most of the work is actually in training a powerful deep master over lots of data that classifies many classes to then transfer some of that knowledge to a specialist net. Luckily, the hard work is already done

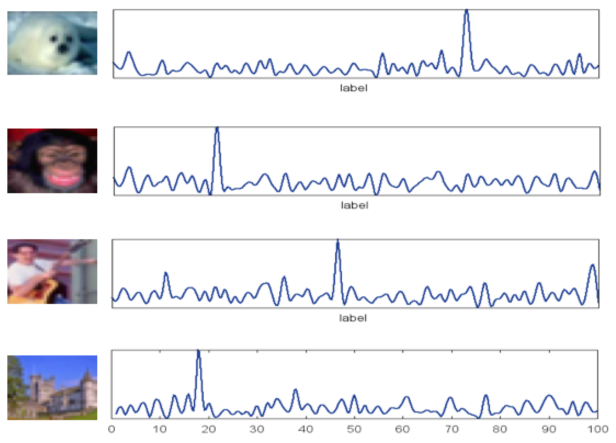

Figure 1: Probability distributions over labels on CIFAR-100

for us, as one can easily download a powerful pretrained master on many of the popular benchmark datasets.

Ideally we would work on the ImageNet dataset, as it has become the most important challenge for image classification with many of highly performing models readily available. But as our computational power available for this project is limited, we chose to work on a smaller dataset that still has a good number of classes: CIFAR-100 [1]. CIFAR-100 is similar as CIFAR-10 (it has 60,000 32x32 images of common objects), but it has 100 classes. This dataset also has the advantage of having classes that have natural clustered structure. The input image will be fed to 9 specialists, each specialists predicting a score for the subset of the classes it is responsible for and a *dusbin* score, representing if that image should not be classified by that specialist.

The performance of our algorithm will be measured by how well that ensemble of 9 specialist is able to reconstruct the final labels over the full 100 classes.

2. Related Work

2.1. Pure compression and knowledge transfer

The more general concept of knowledge transfer has been around for a decade. While it is common practice to average the output of an ensemble of networks that then performs better than the individual nets, Caruana *et al*. [9] showed that one can easily compress this ensemble to the size of an individual component. The goal of the compressed version is to learn the transfer function (or mapping from the input space to the output space) that the ensemble has learned. When unlabeled data is widely available, which is the case for image classification, one can feed the unlabeled data through the ensemble and make the output (raw class score or logits) of the ensemble the label that the compressed model wants to learn. With access to an almost infinite amount of unlabeled data, the compressed model performs almost as well as the ensemble, thus reducing memory and forward pass times by several folds.

That concept has been explored further by Ba *et al*. [13], who's goal was to mimic deep and well engineered neural networks by shallow, but wide, fully connected nets. In order to facilitate learning, they introduced the concepts of learning on logits rather than the probability distribution. This paper presents an interesting approach that questions why deeper networks perform better when trained with traditional supervised back propagation methods, while shallower network can have the same representation power.

2.2. A more general framework: Dark Knowledge

Finally, Hinton *et al*. [2] introduced a more general framework to transfer knowledge efficiently by introducing the concept of *temperature*. Indeed, knowledge distillation can be enhanced by transforming the master's output to a softer probability distribution, by simply dividing the logits by a temperature coefficient before feeding them to a Softmax function (cf. 3.4). The key idea is that by using a higher temperature, the activations of wrong classes are increased, providing more information flowing to the model parameters during backpropagation.

Furthermore, the proposed general distillation framework also emphasises the use of a hybrid loss function which also includes information about the true label. Hinton *et al*. also suggested that training an ensemble of specialists may be an effective way to take advantage of natural parallelism when working with very large datasets and this approach may be able to surpass traditional compression just due to computational constraints when training large models. This suggests that it is possible to transfer class specific knowledge from the master to a specialist only interested in a subset of his classes while still benefiting from the performance boost of training on the whole dataset.

2.3. Using local experts

The idea of using local experts trained on subsets of classes also has been around for a while. Jordan *et al*. present in [10] a framework for training such experts. They are training those networks from scratch, and use a ”gating network”, to indicate which expert to use on a given example.

2.4. Using a dustbin class and unsupervised data

In [14], LeCun *et al*. introduce two concepts that are quite related to our methods, however they have less interaction in our case. First, they advocate for the use of unsupervised learning for regularization. We strongly believe as well, as presented in some sections of this paper, that when possible, unsupervised learning can act as a regularizer. The other shared concept with LeCun *et al.*'s paper is the dustbin. At the scale of one specialist, this dustbin class has the same purpose: regularizing its learning process by showing it images that are outside his domain of expertise. However, on a more global scale of our approach, this dustbin class is used whether or not the label is known for a given image.

2.5. State of the art on CIFAR-100

Because of its low number of training examples per class (400), it is very hard for models to achieve high accuracy on CIFAR-100 (cf 4.1). Current best models achieve 75.72% using Exponential Linear Units [6], 75.7% using Spatially-sparse convolutional layers [7] or 73.61% using Fractional Max-Pooling [8]. All of these use a large amount of data augmentation and unconventional layers.

3. Methods and pipeline

3.1. Global pipeline

We first provide a high-level description of the proposed pipeline to train specialist networks.

1. Decide on a set of clusters in the labels. This can either be done manually or automatically.
2. Get a well trained (ensemble of) network(s) if possible. Otherwise, train one. This is the master network.
3. For each example in the training set, compute the raw scores with the master network.
4. Train a small network for each cluster (this can be **parallelized!**), using a *dark knowledge loss* (cf. 3.4), and the master's scores as targets.
5. For each example in the training set, concatenate all the specialists's scores
6. Learn a simple linear layer on these scores, using the hard labels (cf. 3.6).

All methods were implemented with Torch7 [3] and were trained on a NVIDIA GRID K520 GPU, using an Elastic Compute Cloud instance on Amazon Web Services.

3.2. Creating the soft labels

The first step is to feed the dataset through our master network to create the soft labels. This step can be done only once, by saving the raw scores to disk. After this, the master network is never used (unless training examples are added), and can be discarded, gaining computing time and memory.

3.3. Choosing specialist networks's class subsets

We handpicked the subsets of classes for each specialist, looking at classes that are similar to a human.

A more thorough approach would be using the covariance of the confusion matrix of the master network to see which classes are most confusable, which represent a similarity metric between classes. Figure 2 shows such a covariance matrix, on a network we trained on CIFAR-100 (cf. 5.1). If this is used as a similarity matrix, it shows clusters around its diagonal, as shown in Figure 2.

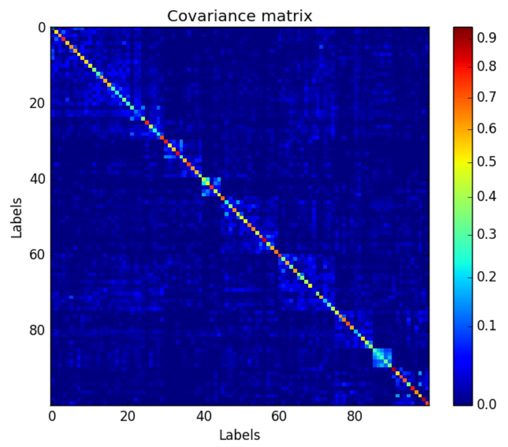

Figure 2: Covariance matrix of our VGGNet's confusion matrix over CIFAR-100

If we use this similarity metric to visualize the classes in a lower dimensional space (using a t-SNE representation, cf. [16] for more details), we can see that it allows clear clusters to appear, although they don't necessarily correspond to the handpicked clusters (cf Figure 3).

Another possibility would be to use a clustering algorithm such as *K-means*, on features extracted from the data. Those features can for example be extracted from the master network's last fully-connected layer.

3.4. The Dark Knowledge loss function

A typical neural networks transforms the raw scores z_i from the final linear layer into a probability distribution q_i for each class i by using a *Softmax* layer: $q_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$. This can be seen as a particular version of a more general transformation:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

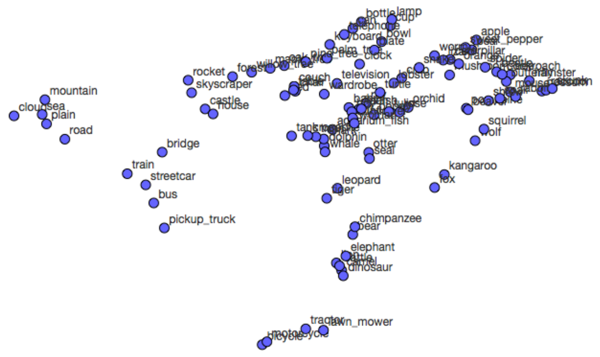

Figure 3: Lower-dimensional representation of CIFAR-100's labels, using t-SNE and our VGGNet's confusion matrix over a test set as a similarity measure (made with [17])

where $T = 1$ for the Softmax function. As explained earlier, raising this temperature hyperparameter is allowing us to soften the probability distribution. The cross entropy loss function for an input x_j and (hard) label y_j is then:

$$Loss_{\text{hard}} = -log(q_{y_j,T=1})$$

Now that we also have soft labels that we want to match, we have to account for a new loss function. We will compare the probability distribution of the master p_i with the distribution of the student q_i at a specific temperature $T > 1$ by using a **Kullback-Leibler divergence** between both distributions:

$$Loss_{\text{soft},T} = -\sum_i p_i log(q_i)$$

where q_i and p_i depend on T according to the above equation. The final loss will actually be a linear combination of both this soft loss and the typical cross entropy $Loss_{\text{hard}}$, parametrized by $0 < \alpha < 1$:

$$Loss_{\alpha,T} = \alpha Loss_{\text{soft},T} + (1-\alpha) Loss_{\text{hard}}$$

If we are working with unlabeled data, we will only use the soft loss, so $\alpha = 1$. On the other hand, $\alpha = 0$ corresponds to vanilla training on the true labels.

An alternative to using a KL divergence on the soft loss, is to simply use an L_2 loss over the the raw scores but Hinton *et al.* [2] showed that the KL divergence on the softened probability distribution approaches an L_2 measure when the temperature is much higher than the magnitude of the logits z_i.

As the soft loss's gradient scales like $\frac{1}{T^2}$, we multiply it by T^2, to ensure consistency when experiment with hyper-parameters, as recommended in [2].

3.5. Specialist dustbin class and biased training set

As opposed to the gating technique proposed in [10], we want each specialist to have the responsibility of deciding whether or not an image is in its given domain of expertise. Thus, each specialist also has a **dustbin** class, corresponding to all the classes it does not specialize in. To create the soft targets for each specialist, we simply took the raw scores of each of the specialist's class and the maximum over all other classes as the dustbin score. This is an arguable choice, as taking the sum of probabilities would have been more mathematically rigorous, but this would have biased all the specialist towards their dustbins, given that the number of classes they specialize in is always much smaller than the total number of classes.

When the number of classes is very large, it makes computational sense to modify the training set to be specialist specific, to have a more balanced training set. We chose to bias it for each specialist such that the proportion of dustbin examples is always around 50%.

3.6. Adjusting for the biased training set and using the specialists for global prediction

However, this latter choice induces a source of error for our ensemble of specialist networks. Each of them is trained on a biased training set, whose balance is very dissimilar to the real one's. Thus their bias need to be corrected accordingly. Moreover, we need a way to leverage the different outputs as well as the dustbins to perform inference on a test set, and find scores for all classes. In conclusion, we want to weight the different outputs according to the level of confidence in each specialist, as well as correct the biases. We can **learn this!**

Thus we propose, after each specialist is trained, to train a final linear classifier, that takes as input the concatenation of the specialists's scores (including the dustbins), and outputs the correct class. This linear layer is trained at the end of the process, in a really short time as both its input and output are quite low-dimensional (this layer does not back-propagate through the specialists).

4. Dataset

4.1. Data

To demonstrate this method, we propose to use the CIFAR-100 dataset, collected by Krizhevsky *et al.* [1], which is very similar to CIFAR-10. It consists of 60,000 images of size 32x32.

The main advantage of this dataset for our particular problem is that it presents a natural hierarchical and clustered labeling structure. It has 20 coarse labels, each of

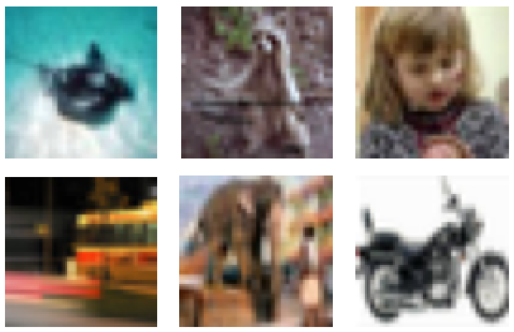

Figure 4: Sample images from CIFAR-100, for classes *ray*, *racoon*, *girl*, *streetcar*, *elephant*, and *motorcycle*

them having 5 fine labels. For example, there are "orchids", "poppies", "roses", "sunflowers", and "tulips" in the "flowers" category, but "baby", "boy", "girl", "man", "woman" in the "people" category.

This structure is making it a perfect candidate for this task as we have clearly clustered classes. Moreover, it will allow us to start by handpicking the specialists with respect to their coarse labels.

We split data into 40,000 images in the training set, 10,000 images as a validation set that we used to cross-validate all of our hyper-parameters and finally a test set of 10,000 images to evaluate the final performance of our specialists.

4.2. Pre-processing

Our dataset is pre-processed in the following way:

- the red-green-blue (RGB) channels are converted to YUV channels, making information easier to extract
- the chrominance channels (U and V) are centered normalized globally (using the training set mean and stantard deviation) the luma channel (Y) is centered and normalized locally (per example)

4.3. Data augmentation

The only form of data augmentation that we used is random flipping: during training, each image that is fed to the network is flipped horizontally with probability $\frac{1}{2}$.

5. Experiments, results and discussion

5.1. Training of the master network

One drawback however of choosing CIFAR-100 is that we were not able to access a pretrained network with state of the art performance. As a result, we decided to train our own master from scratch. We created a VGGNet with $\sim$15M parameters using batch normalization and dropout at some layers (cf. Appendix A for detailed architecture, cf. [18] for more information about batch normalization). In order to spend less time on hyper-parameter tuning, we used some that worked for training a very similar architecture on CIFAR-10 [4]. After a bit of tweaking, and 15h of training, we were able to achieve 67.52%. This compares to a state-of-the-art of 75.72% with $\sim$375M parameters, as explained in 2.5.

Even if this is not the focus of interest, the reader may be interested to know that the key hyper-parameters were a learning rate of 1 (divided by 2 every 25 epochs), a weight regularization of 10^{-4}, optimizing using mini-batch stochastic gradient descent with a Nesterov update (using momentum of 0.9) and batches of 128 images, over 250 epochs (cf. 5 for the master's learning curve).

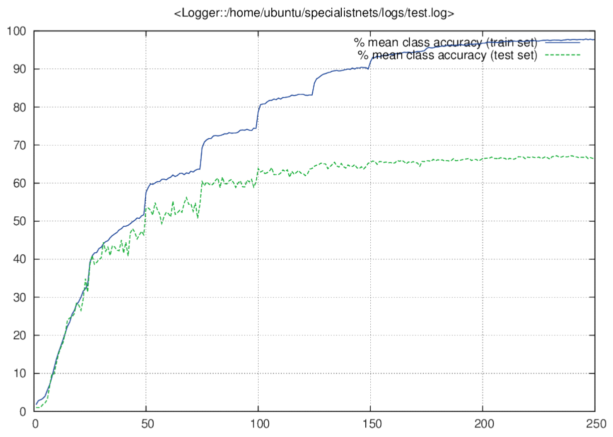

Figure 5: Learning curve for our master VGGNet (blue: training accuracy, green: validation accuracy)

5.2. Pure compression

Our first approach was to experiment with our dark knowledge loss, by performing a simple model compression. We compressed the master model using a VGG-like architecture but decreasing the number of parameters by a factor of 10 (the architecture is not detailed here, as this is not the focus of this paper). Using only the soft labels ($\alpha = 1$), we were able to achieve 58.5%, with no form of regularization (neither dropout or batch normalization). This proves that knowledge distillation indeed acts as a strong regularizer.

However, the same architecture with dropout and batch normalization, trained only with the hard labels ($\alpha = 0$) achieved 63%. This proves that this approach can be limited if the master network did not achieve a really high accuracy, and if we use the same training set for both models: the student learns a prior over the classes, which is built with an over-fitting model, and thus errors are amplified. We believe that compression using the same training set as the master can only be successful if the master is close to 100% accuracy (and thus the function learned by the master is closer to the true function), or by adding a contribution from the vanilla cross-entropy loss (as explained earlier).

5.3. Choosing the specialist clusters and architecture

As we explained earlier, the specialist clusters (later on also called *domains*) can either be handpicked, or automatically determined. For the scope of this paper, we only used handpicked clusters.

We started by working using CIFAR-100's coarse labels as clusters (i.e. 20 clusters of 5 classes each), but this ended up yielding poor results, as each specialist was given too few training examples, and thus each specialist preferred predicting each example as dustbin. Thus, we quickly moved to choosing less clusters. We finally decided on 9 clusters, which are probably sub-optimal but were sufficient for the scope of this paper, although this should be explored (cf. Appendix B for the list of clusters).

We explored different architecture for the specialists, keeping in mind the trade-off between achieving an efficient compression (both for memory and time reasons), but keeping a reasonable representation capacity. We finally decided on the architecture presented in Appendix C, where each specialist has about 170k parameters.

We can get a first baseline for our approach, which is the representation capacity of this architecture. We trained this architecture on all classes, using the vanilla hard loss, to show that it is not sufficient all by itself to achieve a decent accuracy on CIFAR-100. When training it with similar hyper-parameters as the master network, we get a test accuracy of 46.53%, and need to use spatial batch normalization for regularization, considerably slowing the process.

5.4. Training specialists with hard labels only

A first interesting thing to try is training the specialist in a more conventional way, without only the master network. They are first trained with binary labels as targets, before training our linear classifier for ensemble predictions.

As expected, they over-fit a lot, and the final accuracy over the 100 classes is quite low (about 44.5%), using regularization methods such as dropout and batch normalization.

5.5. Training the specialists

The next step is to actually train the specialists, using our dark knowledge loss and the soft targets produced by the master saved on disk. Also they can be trained in parallel, for simplicity we decided to train them one after another. It's interesting to see that for one specialist, the training time is about 70 times faster than the master (this depends on the domain size). This still holds for training the 9 specialists as they can be trained in parallel.

Again, we used mini-batch stochastic gradient descent with Nesterov update (using a momentum of 0.9), and batches of 128 images. It seems that temperatures around 10 often work better.

5.5.1 Specialist-specific hyper-parameters

Since they all have very different images in their domains, and have different output dimensions, we decided to have different hyper-parameters for each specialist. The significant speed-up presented above allowed us to conduct this search in a reasonable time, whereas searching hyper-parameters for the master network is much longer. The three hyper-parameters on which this search was conducted were learning rate, the temperature, and α (the relative influence of soft and hard loss). There seemed to be no specific relationship between the best hyper-parameters, and the domain size, except maybe for the learning rate, where greater rates seemed to work better with larger domains. We believe that the optimal hyper-parameters depend more on the data itself and on its distribution in the clusters than on domain size. The hyper-parameters that worked best are available in Appendix D.

Figure 6 shows a typical learning curve for a specialist with a good choice of hyper-parameters.

After training the final linear classifier, we reached a global test accuracy for the specialists of **54.42%**. This is still lower than the purely compressed model, but the architecture has way less parameters, and the ultimate goal of specialists is also to be able to extract a subset of the master knowledge (some of the specialists reached 85% test accuracy).

5.5.2 Speed-up and memory-gain

With 9 specialists and the previous architecture, the total number of parameters falls down to about 2 millions (counting all the specialists), gaining a factor of over 10 compared to our master, and to more than 100 compared to the state-of-the art networks.

Training the specialists also allows us to observe to significant speed-up allowed by this approach. Since they can

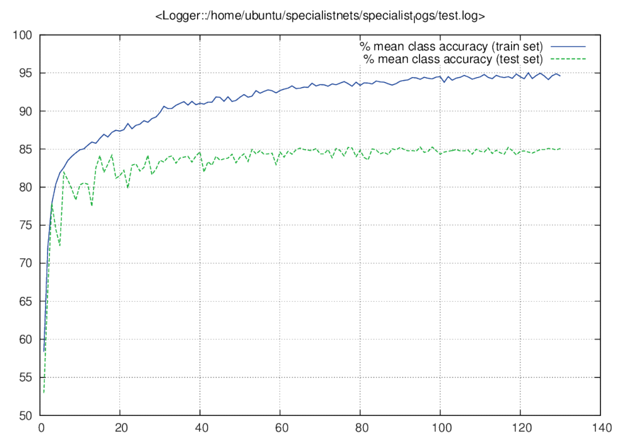

Figure 6: Learning curve for the *Outdoor* specialist (blue: training accuracy, green: validation accuracy)

be trained in parallel, the whole ensemble was trained about **70 times** faster than our master network (allowing for efficient hyper-parameter tuning and architecture exploration).

Moreover, after training, the forward-pass for the ensemble is **63 times** faster than our master network, making those networks much more easily embedded on portable devices.

5.6. Initializing the specialists weights

So far the problem of weight initialization for our small specialists has not been addressed. In all of our training processes, we have been using a *MSR* scheme [15], where the weights for a given layer i are drawn from the following normal law, where H and W are the filter's dimensions, and F is the number of filters on a given layer:

$$w_i \sim \mathcal{N}\left(0, \sqrt{\frac{1}{W_i \times H_i \times F_i}}\right)$$

However, we can find smarter way of initializing the specialists. Since such a small architecture is so fast to train, we can start by training a compressed model that has the exact same architecture (but 100 outputs). We can then use the weights of this model as an initialization for our specialists (except, obviously, the last fully-connected layer). This did not yield an increase in global accuracy, but allowed us to gain considerable training time for the specialists, which started at an accuracy usually reached after about 40 epochs.

5.7. Adding an overview model

Since we trained this reduced model with 100 classes, we can just add it to the ensemble! Therefore we have 10 ”specialists”, 9 of which are actual specialists trained on a subset of classes, and the last one being this ”overview” model, which is just as small as the other ones, and helps guiding the overall ensemble towards the correct class.

Adding this compressed model allowed a gain of almost 1%, the overall ensemble reaching a test accuracy of **55.32%**.

5.8. Unsupervised learning

Since we are only using the master's prediction for training, we can actually use unlabeled data to train the specialists. This can have a couple of issues, such as the fact that we are using the same training set for the master and the specialists, and the limited size of CIFAR-100.

Thus we tried to train the specialist networks through unsupervised learning, using data from CIFAR-10 (in theory, this increases the potential size of the training set to all available images!). With weights initialization using our pre-trained small network, unsupervised learning with 10,000 examples yielded an accuracy of 53.12%, whereas this got an accuracy of 46.01% (with far less over-fitting) without such initialization, with 50,000 examples. Note that in the latter case, this accuracy is reached **without the specialists ever seeing any image from CIFAR-100's classes!** Even if the performance is lower, this is an impressive fact and truly explains the name of *dark* knowledge.

For timing constraints, we weren't able to explore unsupervised learning further. Based on the learning curves which did not reach a plateau when the experiments were stopped, pure unsupervised learning could have yielded better results with more training time, which is encouraging for further research.

6. Conclusion and future work

In this paper we have developed a framework for building an ensemble of specialists, trained from a master network through knowledge distillation, and performing prediction with this ensemble with an additional linear classifier. We have also explored various ways of potentially improving the results, through a better weight initialization, and by adding an overview network. This approach can be particularly efficient if only a subset of the master's knowledge needs to be extracted.

Our best model was using our specialists, where the initialized with a small model that was pre-trained by knowledge distillation on all classes, and combining them with this particular model for prediction. The ensemble performed best when choosing specialist-specific hyper-parameters, and achieved a test accuracy of **55.32%** on CIFAR-100. This is still lower that our master network, but with significant time and memory gain.

Some future work should be planned to improve this approach.

First, as we stated in the paper, the clusters were handpicked, and did not necessarily correspond to confusion clusters. Thus they should be chosen with more automated method, such as clustering on the confusion covariance matrix.

This approach should also be done using a master that is closer to state-of-the-art result, by getting a model from the top-performing paper on CIFAR-100, or switching to a different data set such as ImageNet.

Finally, the use of unlabeled data should be explored, by using a lot more images (in this case, from the 80M Tiny Images data set), and by trying training the models with a mixture of labeled and unlabeled data. We believe the ensemble's behavior when only trained with unlabeled data is encouraging for further research.

Appendix A: Master network architecture

Each convolutional layer is followed by a Spatial Batch Normalization layer, and a ReLU activation.

64 3X3 CONV + 0.5 DROPOUT
64 3X3 CONV
2x2 MAX-POOL
128 3X3 CONV + 0.4 DROPOUT
128 3X3 CONV
2x2 MAX-POOL
256 3X3 CONV + 0.4 DROPOUT
256 3X3 CONV + 0.4 DROPOUT
256 3X3 CONV
2x2 MAX-POOL
512 3X3 CONV + 0.4 DROPOUT
512 3X3 CONV + 0.4 DROPOUT
512 3X3 CONV
2x2 MAX-POOL
512 3X3 CONV + 0.4 DROPOUT
512 3X3 CONV + 0.4 DROPOUT
512 3X3 CONV
2x2 MAX-POOL + 0.4 DROPOUT
512x512 FULLY-CONNECTED
BATCH NORMALIZATION - ReLU
0.5 DROPOUT
512x100 FULLY-CONNECTED
SOFTMAX

Appendix B: Handpicked specialist domains

specialist	classes
Mammals	hamster, mouse, rabbit, shrew, squirrel, bear, leopard, lion, tiger, wolf, camel, cattle, fox, chimpanzee, elephant, kangaroo, porcupine, possum, raccoon, skunk
Aquatic Animals	fish, flatfish, ray, shark, seal trout, beaver, dolphin, otter, whale
Other Animals	crab, lobster, snail, spider, worm, crocodile, dinosaur, lizard, snake, turtle
Flora	orchids, poppies, roses, sunflowers, tulips, apples, mushrooms, oranges, pears, peppers, maple, oak, palm, pine, willow
Home	bed, chair, table, wardrobe clock, keyboard, lamp, couch, telephone, television
Vehicles	bicycle, bus, motorcycle, pickup truck, train, lawn-mower, rocket, streetcar, tank, tractor
People	baby, boy, girl, man, woman
Outdoor	cloud, forest, mountain, plain, sean, bridge, castle, house, road, skyscraper
Objects	bottles, bowls, cans, cups, plates

Appendix C: Specialist networks architecture

Each convolutional layer is followed by a ReLU activation.

32 3X3 CONV
2x2 MAX-POOL
32 3X3 CONV
2x2 MAX-POOL
64 3X3 CONV + 0.4 DROPOUT
2x2 MAX-POOL
128 3X3 CONV + 0.4 DROPOUT
2x2 MAX-POOL
128 3X3 CONV + 0.4 DROPOUT
2x2 MAX-POOL
512x128 FULLY-CONNECTED
BATCH NORMALIZATION - ReLU
128x(#classes) FULLY-CONNECTED
SOFTMAX

Appendix D: Best hyper-parameters for the various specialists

specialist	T	α	learning rate
Mammals	4.8	0.998	6.6
Aquatic Animals	7.2	0.985	1.4
Flora	13	0.999	11.9
Objects	7	0.999	1.5
Home	27	0.985	4.6
Other Animals	6.6	0.991	8.3
Outdoor	8	0.982	5.57
People	28	0.998	1.47
Vehicles	14.3	0.999	4.18

References

[1] Krizhevsky, Alex, and Geoffrey Hinton. ”Learning multiple layers of features from tiny images.” (2009).

[2] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. ”Distilling the knowledge in a neural network.” *arXiv preprint arXiv:1503.02531* (2015).

[3] Collobert, Ronan, Koray Kavukcuoglu, and Clement Farabet. ”Torch7: A matlab-like environment for machine learning.” *BigLearn, NIPS Workshop.* No. EPFL-CONF-192376. 2011.

[4] Zagoruyko S., 92.45% on CIFAR-10 in Torch, *Torch Blog*, 2015.

[5] Lin, Min, Qiang Chen, and Shuicheng Yan. ”Network in network.” *arXiv preprint arXiv:1312.4400* (2013).

[6] Clevert, Djork-Arn, Thomas Unterthiner, and Sepp Hochreiter. ”Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).” *arXiv preprint arXiv:1511.07289* (2015).

[7] Graham, Benjamin. ”Spatially-sparse convolutional neural networks.” *arXiv preprint arXiv:1409.6070* (2014).

[8] Graham, Benjamin. ”Fractional max-pooling.” *arXiv preprint arXiv:1412.6071* (2014).

[9] Bucilua, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil. ”Model compression.” *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2006.

[10] Jacobs, Robert A., et al. ”Adaptive mixtures of local experts.” *Neural computation* 3.1 (1991): 79-87.

[11] Romero, Adriana, et al. ”Fitnets: Hints for thin deep nets.” *arXiv preprint arXiv:1412.6550* (2014).

[12] Balan, Anoop Korattikara, et al. ”Bayesian dark knowledge.” *Advances in Neural Information Processing Systems.* 2015.

[13] Ba, Jimmy, and Rich Caruana. ”Do deep nets really need to be deep?.” *Advances in neural information processing systems.* 2014.

[14] Zhang, Xiang, and Yann LeCun. ”Universum Prescription: Regularization using Unlabeled Data.” *arXiv preprint arXiv:1511.03719* (2015).

[15] He, Kaiming, et al. ”Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.” *Proceedings of the IEEE International Conference on Computer Vision.* 2015.

[16] Van der Maaten, Laurens, and Geoffrey Hinton. ”Visualizing data using t-SNE.” *Journal of Machine Learning Research* 9.2579-2605 (2008): 85.

[17] Karpathy A., tSNEJS, https://github.com/karpathy/tsnejs, 2015.

[18] Ioffe, Sergey, and Christian Szegedy. ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.” *arXiv preprint arXiv:1502.03167* (2015).