CSCI 2470/Capstone Final Project Report: Little Label Learners: Gradual Supervision Training Regime for Learning Novel Classes

Team members: Winston Li, Marcel Mateos Salles, John Ryan Byers
TA name: Preetish Juneja . Brown University

Abstract

Modern neural networks often use vast volumes of data, far more than what a human needs, to learn to recognize common objects. Plus, these models often suffer from catastrophic forgetting during continual learning tasks, whereas humans can learn nearly unlimited numbers of novel conceptual classes.

With Little Label Learners, we present a gradual supervised learning routine that uses increasing numbers of class-representation and labeled-data over epochs to mimic the data diet of infants. We demonstrate that our model is able to learn new classes effectively, comparable to fully-supervised routines using significantly more labeled images and datasets that contain all labels of interest.

1. Introduction

Unlike neural networks, infants have limited access to visuals and labels. Self- and semi-supervised learning seek to address this constraint. The former uses no labels, and the latter uses limited amounts to define a latent manifold.

We explored an intermediate: gradual supervision. Starting with pure self-supervision, we added in more labels over epochs. This defined more informed manifolds than pure self-supervision while using less labels than static semi-supervision models.

Using Tensorflow [1], we begin with a purely self-supervision regime working with a limited subset of the CIFAR-10 dataset [2]; for example, we only use data corresponding to 7 classes. As epochs increase, we gradually "leak" in labeled images to represent a transition to a semi-supervision learning regime. This provides "names" to our learned latent clusters.

Similarly at later epochs, we introduce data from new classes. Thus, the model must not only learn to predict new labels in a zero-shot fashion but also remember them as new possible prediction label choices. At the final epochs, we consider all 10 classes.

2. Methodology

Overall, our model first embeds both labeled and unlabeled images using an encoder. In this latent space, we can then optionally use a pseudo-classifier objective to further train the encoder. Next, images are passed through a projection head and then used in a contrastive loss objective. Finally, a linear probe projects the latent embeddings into classification space, allowing for CCE between predicted labels and true labels. This loss can either be used to only update the linear probe, or also the encoder if a semi-supervised method is desired.

2.1. Related Works

Self- and semi-supervised methods have been popular research topics in the deep learning field. For our project, we referred primarily to two papers: *SimCLR* by Chen et al. [3] and *Local Label Propagation* by Zhuang et al. [4].

In *SimCLR*, the authors present a contrastive-learning objective via image augmentations. For each image, two different series of augmentations are applied followed by an encoder layer. The contrastive-learning objective requires the encoder to minimize the distance between pairs of augmentations that came from the same image while maximizing the dissimilarity between non-same pairs of augmented images. Intuitively, this requires the model to learn deeper semantic meanings that are robust to augmentations; hence, a grayscale dog is the same as a rotated dog. Since the model only requires images rather than labels, this is a purely self-supervised method.

To validate their model, the authors used a "linear probe", which was essentially a simple dense layer linearly projecting embeddings into classification space (i.e. one with the same number dimensions as there are classes). The linear probe predicted an image's class from its encoded representation, and the predictions were used to train the linear probe.

The *SimCLR* training method inspired us to use an augmentation-contrasted learning objective. We also integrated the linear probe into the encoder training regime by using the former's classification loss to update the latter's weights. This made our model semi-supervised.

In *Local Label Propagation*, Zhuang et al. introduce a semi-supervised method for using labeled data to first propagate "pseudo-labels" to unlabeled data. Then, the authors used a categorical cross-entropy loss function between predicted and propagated pseudo labels, in addition to a global aggregation metric, to update their encoder.

This paper inspired several of our pseudo/meta-labeling experiments for updating our encoder outside the linear probe. However, most of our experiments understood "pseudo" instead as with respect to labeled points, whereas Zhuang et al. understood "pseudolabels" with respect to the unlabeled points.

More specifically, the Zhuang et al.'s loss value comes from the comparison between pseudolabeled *unlabeled* points and model-predicted *unlabeled* points. In this sense, their architecture sets the model-prediction has the "ground-truth" on which encoder quality is compared to. Conversely, our algorithm's loss value compares pseudolabeled *labeled* and the original *labeled* points. Thus, our architecture sets the actual *labeled* points as "ground-truth" and instead uses *unlabeled* points as a way to modify *labeled* points. We believe this is a much more intuitive understanding of a "ground-truth".

2.2. Dataset

For this project, we knew that we would be constrained in time and processing power. This led us to discard the possibility of using ImageNet [5]. However, we still wanted to use a reliable dataset with a decent amount of data across multiple categories in order to be able to train the model with a certain amount and slowly introduce more. These constraints led us to choose CIFAR-10 as our dataset for the project. This dataset consists of 50,000 32x32 colored training images and 10,000 testing images. It is made up of 10 categories, each containing 6000 images. It only takes up about 132 MB of storage and is easily accessible through Keras [6].

2.3. Preprocessing

Our preprocessing was crucial to the success of our model. First, we normalized all the data. Then, we created a class object that would allow for us to divide the data into different subsets, based on what our model would needed for its training. We called this the `Dataloader` class. With this class we could decide how much labeled data the model would have access to while also being able to choose the number of categories represented in the data, labeled or unlabeled. This was crucial to the gradual learning aspect of our model. Preprocessing and the `Dataloader` object allowed us to easily divide the data into different groups whenever the need arose in an easy and accessible manner.

2.4. Architecture

Our model can be split into three major components: the encoder, the pseudo-classifier, and the linear probe. The encoder was responsible for encoding images into a rich embedding space. The pseudo-classifier was responsible for using labeled images to separately train the encoder in addition to the encoder's contrastive learning objective. Lastly, the linear probe provided both a running metric of our latent space quality as well as an opportunity for updating the encoder.

2.4.1 Encoder

The encoder was comprised of four `Conv2D` layers with `LeakyReLU` activations in-between followed by a `Dense` layer. For our contrastive learning objective, encoded images were also passed through a projection head, which featured two more `Dense` layers with a `LeakyReLU` in-between.

2.4.2 Pseudo-Classifier

We experimented with several pseudo-classifier components, but they were all conceptually similar; for example, they all began by encoding labeled and unlabeled images. One experiment involved explicitly dividing labeled encodings by class. Then, for each unlabeled point, a similarity metric was calculated with respect to all other labeled points by class. This tensor was then softmaxed across classes to obtain a "pseudo-label" for each unlabeled point. In other words, for each unlabeled encoding, we calculated a distribution across labels whose probability density function was related to its average similarity per classes of labeled encodings.

For each class of labeled encodings, we calculated the weighted average similarity with respect to all other unlabeled encodings. In other words, we used the "pseudo-labeled" encodings to predict the class of our labeled encodings. We took the categorical cross entropy between the true labels and pseudo-classified labels to obtain a notion of "pseudo-loss", which was then used to update the encoder.

The above approach 2.4.2 was greatly slowed down by first having to split the images by class before separately encoding them. This also left us several losses per class of labeled encodings. One implementation would be to have multiple gradient updates; alternatively, a quicker implementation would accumulate losses weighted by the class' proportional representation.

Hence, further experiments did not require the initial class-based splitting step. Instead, we assumed the labeled images implicitly carried label information. Thus, as before, we first calculated the similarity between labeled and unlabeled points. We then weighed the unlabeled points by the similarity. Finally, for each labeled point, we found the weighted average similarity to all unlabeled points. Conceptually, this meant first calculating the importance of each unlabeled encoding with respect to each labeled encoding. Then, the unlabeled encodings are weighed by these similarities. Finally, the points are "influenced" by each unlabeled point by a degree specified by the similarities, thus generating "meta" points 2.4.2

One experiment then used a separate meta-linear-probe to classify the meta points. The categorical cross entropy loss that resulted between meta-classified and true-labels were used to update the encoder. We found that training this meta-linear-probe at the same time as the encoder to be inconsistent and unstable.

To remedy this, our last experiment simply used the averaged, log Euclidean distance between each labeled encoding and its meta-encoding. This led to comparatively more stable learning.

Algorithm 1 Pseudo-Label Classification Algorithm

```
$\hat{U} = enc(U)$

/* Split labeled images by class */
for $c \in C$ do
    $\hat{L}_c = [enc(L_x) | x = c, x \in X]$
end for

/* Pseudo-label each unlabeled point
*/
for $\hat{x} \in X$ do
    $w_x = softmax[avg\_sim(U_x, L_c), c \in C]$
end for

/* Reclassify each labeled point
using pseudo-labeled points */
$\mathcal{L}_{pseudo-classified} = 0$
for $\hat{c} \in C$ do
    $\tilde{l}_c = softmax[avg\_sim(w_x * U_x, L_c, x \in X]$
    $\mathcal{L} = \mathcal{L} + CCE(\tilde{l}_c, l_c)$
end for
```

In general, we found that the pseudo-classification domain did *not* significantly affect the linear probe's classification accuracy. However, the final experiment *did* improve our contrastive performance. This may suggest that our linear probe is too simple and thus weak to fully take advantage of our latent space representations.

2.4.3 Linear Probe

The linear probe comprised simply a `Dense` layer without an activation. This was meant to represent a linear regression on top of the latent-space encodings of input images to project into classification space.

Algorithm 2 Meta-Label Distance Algorithm

```
$\hat{U} = enc(U)$
$\hat{L} = enc(L)$

/* Similarity between each unlabeled
encoding and each labeled encoding */
$S(\hat{U}, \hat{L}) = \hat{U} \cdot \hat{L}^T$
$S = \text{softmax}(S)$

/* Weigh unlabeled points by
similarity WRT labeled points */
$W = einsum(`ij, ik \to jik', S, \hat{U})$

/* For each labeled point, find
weighted average to all unlabeled
points, thus creating a meta point */
$\tilde{L} = einsum(`ijk, ik \to ik', W, \hat{L})$

/* Find average log
Euclidean-Distance between each
labeled point and its meta
counterpart */
$\mathcal{L} = \frac{1}{N} \sum_N \log ||\tilde{L}_n - \hat{L}_n||_2$
```

2.5. Gradually Supervised Training Regime

To mimic the variable data diet present during early infancy, we used a epoch-scheduled callback to vary the number of classes represented by the data, the proportion of data that were labeled versus unlabeled, as well as a separate learning rate for the encoder from the linear probe. Thus, our `ScheduledSubsetCallback` class can be split into three major functions: class-based subsetting, label splitting, and learning-rate updating.

In class-based subsetting, we sought to increase the diversity of classes represented as epochs increased. This was meant to reflect an infant seeing novel objects. For our model, we simply tied subset size to the current epoch, such that on epoch n, our subset size had n out of ten total classes represented. Crucially, class-based subsetting also represented the model's ability to learn completely new classes of data.

In label splitting, we wanted to represent an infant's growing capacity to semantically associate names with ideas. Such names represent labels, named ideas labeled images, and un-named ideas unlabeled images. For our model, we tied the split-rate to the current epoch, such that on epoch n, $\frac{n}{\text{total number of epochs}}$ of the data were labeled.

In updating the learning rates, we reasoned that both the encoder and the linear probe such become less plastic at later epochs, the encoder especially so. For our model, we exponentially decayed the learning rate with respect to the current epoch n with learning rate $\eta = \eta_{ceiling}(0.1)^{n/500}$.

Note, we chose to have the encoder's learning rate be significantly smaller than that of the linear probe to represent the implasiticity of the more fundamental encoder. For example, the encoder extracts "axiomatic" beliefs that would be hard to change!

2.6. Model Types

For our project, we worked primarily with three models: baseline, purely self-supervised, and gradually supervised.

In the baseline model, the architecture was composed of the encoder layer followed by the linear probe. Importantly, there was no contrastive learning step; instead, images were directly encoded and then classified in a single CCE calculation via linear probe.

In the purely self-supervised model, the architecture was also an encoder layer followed by the linear probe. However unlike the baseline, in its train step, there were two distinct gradient tape steps: a contrastive learning step where the encoder was trained, and a classification step that used an untrainable encoder and trainable linear probe. While the linear probe could have also been trained separately from the encoder (i.e. after running the model only with contrastive loss), training the linear probe at the same time gave a live, indirect measure of our model's classification over epochs.

In the gradually supervised model, the architecture featured an encoder layer followed by an optional pseudo-classification layer before concluding with a linear probe layer. There were several distinct gradient tape operations: the contrastive learning step for training the encoder, the pseudo-classification step also for training the encoder, and the classification step for training the linear probe and optionally the encoder. This model also featured the scheduled callback, which is discussed in section 2.5.

3. Results

To analyze how well our model performed, we conducted several experiments. First, we used dimensionality reduction methods to visualize our embedding space. Second, we demonstrated that our model is able to successfully learn new labels in an iterative manner. Third, we compared our model to a simple baseline and found that we reached our base and target goals set out in our original proposal.

3.1. Encoder Cluster Quality

Beyond using a linear probe to measure the quality of the encoder's latent space, we also reasoned that a good latent space should feature distinct features for different classes. As our latent space had dimensions $\mathbb{R}^{128}$, we first had to reduce the dimensions down to $\mathbb{R}^2$. This was done in two separate ways: selecting the top two principal components via PCA and using UMAP [7].

In general, we found from the PCA that images of the same class successfully clustered together, but the clusters overall were rather mixed and poorly separated. A similar result can be seen in the UMAP representation as seen in Figure 1. We reasoned that this may be partially attributable to our weak encoder architecture. Nevertheless, a major motivating point of our project is that we wanted a very low-weight architecture. Further experiments may consider more complicated encoder structures, such as ResNets [8] and ViTs [9].

3.2. Learning New Labels

Our model has the ability to learn new labels with an embedding space that contains numerous clusters. This clustering ability allows the model to incorporate new labels as it trains. To achieve the highest accuracy on all 10 classes, we ran experiments with the gradual introduction of labels. We found that starting with 1 class and introducing 1 new label every epoch achieves an accuracy of 47.87% on all ten classes. Accuracy generally increased when more labels were introduced earlier in the training regimens. The highest accuracy for our novel model was when all 10 class labels were introduced in the first epoch. With this model, we achieved 51.53% accuracy.

3.3. Comparisons to Baseline and Self-Supervised Models

Compared to the fully supervised Baseline model, our gradual-learning method proved to be efficient. While final accuracy was was worse than Baseline, our model used significantly less data and had a more challenging task of having to learn new classes. Considering some napkin math, the Baseline model had access to 10 classes for 20 epochs and seeing 50 all of the labeled data, leading to approximately $5000\ images/class \times 10\ classes \times 20\ epochs = 1,000,000\ images$. Keeping split rate constant, our model would have seen $\left(\sum_{i=1}^{10} 5000\ images/class \times 0.5\ split \times i\ classes\right) + 5000\ images/class \times 0.5\ split \times 10\ classes \times 10\ epochs = 387,500\ images$. Our model basically had a third of the labeled information available! Even if we consider if Baseline model having a split rate of 0.5, the fact that the Baseline model cannot use unlabeled data suggests our model is more economic in its data compatibility, and the fact that it still uses less labeled data.

Furthermore, our model performed significantly better than the purely self-supervised model, which suggests that our model successfully incorporated labeled information in an effective manner.

4. Challenges

4.1. Hardware and Software Compatibility

One challenge that we faced was using Tensorflow on Oscar CCV. Our team was familiar with using Oscar, but we

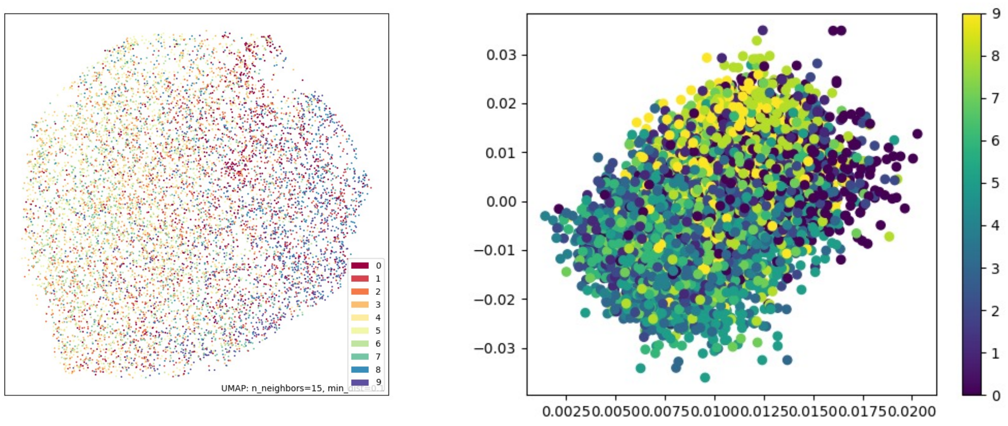

Figure 1. *Left*: 2 dimension UMAP representation of our embedding space after training. Each color corresponds to a different label. *Right*: 2 dimension PCA representation of our embedding space after training. Each color corresponds to a different label.

Model	Learning	Train/Test	Max Validation (%)
Baseline	Fully	10/10	59.18
		7/10	43.11
		7/7	59.59
		10/7	54.97
SimCLR	Self	10/10	31.27
		7/10	25.30
		7/7	37.61
		10/7	32

Table 1. Comparison of baseline model against a simple SimCLR implementation. Train/Test refers to number of classes represented in train/test datasets, respectively.

had trouble activating the GPU support with Tensorflow. To fix this issue, we used the Oscar recommended approach of activating an apptainer. This consolidated the python environment into a docker container that could be run with GPU support.

4.2. Data Handling

Another challenge we faced was converting the CIFAR-10 dataset into a dataset that could be used with self supervision and gradual supervision. The original CIFAR-10 dataset contained labels for every image. However, for our model we needed to have some images come with labels while others came without. In addition, we needed more flexibility with each batch. With our gradual supervision pipeline we wanted to slowly introduce more and more labeled data to our model. To solve these challenges we implemented a Dataloader class using python object oriented programming. Our Dataloader had the functionality for batching together unlabeled and labeled data. The Dataloader also had the flexibility to modify the proportion of labeled and unlabeled data in each epoch of training. Overall, the Dataloader class was an important step in preprocessing to get the baseline and our original model to train.

Another challenge we encountered was the image resolution of our dataset CIFAR-10. This dataset has images of dimension 32 by 32 with a total of 1024 pixels. This low resolution hindered the encoder because there was a limit to the number of features to extract. This made it difficult to surpass the accuracy of the baseline model. We attempted to explore this issue by conducting testing on an additional dataset. We found that the dataset MiniImageNet could suffice with images of dimension 128 by 128. We were able to download this dataset, but we did not have time to process it into the format necessary for our model. This would have necessitated constructing an additional Dataloader class to process this data with a new format.

4.3. Encoder Training Regimes

Finally, preliminary testing with our encoder training regimes showed that while contrastive performance increased significantly (at times 5% or better), it did not significantly improve classification performance. We believe that this may be attributable to several factors: first, the weakness of our encoder; second, our definitions of point-similarities. More specifically on the second point, we are currently using dot-products. An immediate next-step would be to convert these into cosine similarities, which can be done simply by first normalizing across batches prior to multiplication. Alternatively and particularly for our class-naive meta-classifier methods, we currently generate meta points via similarity between labeled and pseudolabeled points. However, it may be more accurate to reimagine meta-points as labeled points being dragged in the direction of pseudolabeled points; this

Model	Learning Type	Class Subset Range	Split Rate Range	Max Validation (%)
Baseline	Full	[10, 10]	[1,1]	59.18
SimCLR	Self	[10, 10]	[0.5, 0.5]	31.27
Novel Model	Gradual	[10, 10]	[0.01, 0.9]	51.53
		[9, 10]		49.49
		[8, 10]		49.73
		[7, 10]		47.66
		[6, 10]		48.66
		[5, 10]		49.27
		[4, 10]		48.49
		[3, 10]		48.64
		[2, 10]		47.17
		[1, 10]		47.87

Table 2. The gradually supervised model is able to learn novel classes without significant decreases in accuracy. Ranges are given as `[floor_val, ceil_val]`, inclusive.

would require a vector subtraction, which may slightly increase computation costs.

5. Reflection

How do you feel your project ultimately turned out? How did you do relative to your base/target/stretch goals?

Overall, we are very proud of how our project turned out. When planning our project, we set certain base, target, and stretch goals that we feel we either reached or were really close to reaching. For the base goal, we feel like the embedding space our model ended up creating was fairly solid. Images of the same class indeed tended to cluster together as visualized in our dimensionally reduced latent spaces.

For the target goal, we feel like we were able to reach it. We wanted to be able to beat or come close to a purely self-supervised model that used basically the same architecture. We are happy to be able to say that our gradually supervised model was able to beat the accuracy of the fully self-supervised mode. Additionally, we feel like the different accuracies achieved by our gradually supervised model of around 50% is close to that of 59% from the fully-supervised baseline model. The definition of what is close might differ by reader, but we felt this was solid considering the fact that the fully supervised model had access to the whole dataset for all the epochs to train on.

We feel like we were fully able to reach our stretch goal. Our model has the capability to be introduced to new labels as the epochs progress. For example, in the first epoch, we would begin its training with the data and labels corresponding to 5 different categories. Then, over the course of the remaining epochs, we would introduce the data and label corresponding to another category and so on until all the categories were introduced. Judging by how our accuracy was higher for some cases when we began with less labels, we feel like our model did a pretty good job of learning these new categories and being able to predict them.

Did your model work out the way you expected it to?

We feel like our model did work out the way we expected, if anything, we were a little surprised. To our delight, the model was able to learn new classes, the way that a child does. We did not really expect it to be able to beat the fully-supervised model, but were surprised by the fact that it was able to be within 10% from that model. This made us very hopeful for this approach.

How did your approach change over time? What kind of pivots did you make, if any? Would you have done differently if you could do your project over again?

Our overall approach did not change that much over time. We spent a lot of effort in the planning stages to ensure that our project was feasible. This included choosing a dataset that we knew we could handle both in terms of storage and compute resources. We also spent time understanding our research papers to outline the scope of the project. This enabled us to stay on track while working on the project, gradually building more and more functionality. In the implementation stages of the project, we had to solve some problems to get our data into the `Dataloader`. However, this did not lead to any major pivots in the project.

What do you think you can further improve on if you had more time?

One thing that we could improve on if we had more time is developing the functionality of training on another dataset. As we detailed above, we would have liked to train on MiniImageNet which is a subset of ImageNet. MiniImageNet provides images of higher resolution with an average image size of 128 by 128. However, we ran into challenges incorporating this dataset into our training pipeline. This was mainly due to our training pipeline requiring batching of labeled and unlabeled data. In order to achieve this functionality much additional preprocessing would have to occur. This was more difficult with the MiniImageNet which was orders of magnitude larger than CIFAR-10. However, using this additional dataset would have helped solidify our results and possibly corroborate our findings on two different datasets.

If we had more time we would have also liked to incorporate the VGG16 architecture into our encoder. Using transfer learning the VGG16 architecture and weights without the softmax layer might have produced a richer embedding space. However, since we did have time constraints, we started with a simple convolutional encoder architecture that we trained from scratch.

A final addition that we would have liked to include with more time is a hyperparameter tuning pipeline. Currently all of our parameters are modified within the body of the code. To make our repository more polished with more time, we would make all parameters inputs through the command line when running the model. This would allow us to run multiple model variations in a row through Oscar batch scripts.

What are your biggest takeaways from this project/what did you learn?

Our biggest takeaway from the project is that semi-supervision can provide an excellent alternative for fully supervised learning. We have shown that a semi-supervised models can achieve similar results in accuracy to fully supervised models with less labels and training. This is important as deep learning models reach domains that have less data. Some tasks such as classification have access to internet scale data with something as simple as a google search. However, other task such as detecting rare diseases on x-ray or medical scans do not have an abundance of data. In order to classify these diseases more accurately, semi-supervision models might be necessary.

Another takeaway is that using insights from a biological perspective can be extremely useful in creating novel deep learning model architectures. We based our gradual supervision off of the type of learning a human child might encounter. This learning strategy introduces lots of different objects some of them without labels. This is akin to a child seeing different things and recognizing their differences but not knowing their names. We then slowly introduce the labels to the model after it has created an embedding space with differences. Just like a child being told different object names, the model learns the labels to the embedding clusters it has found. This biological foundation was the inspiration behind our learning architecture and pipeline. The world of psychology could most likely be further used to develop further advancements in the field of deep learning.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1

[2] A. Krizhevsky. Learning multiple layers of features from tiny images. 1

[3] T. Chen et al. A simple framework for contrastive learning of visual representations. (arXiv:2002.05709), June 2020. arXiv:2002.05709 [cs, stat]. 1

[4] C. Zhuang et al. Local label propagation for large-scale semi-supervised learning. (arXiv:1905.11581), May 2019. arXiv:1905.11581 [cs]. 1

[5] J. Deng et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009. 2

[6] F. Chollet et al. Keras. `https://keras.io`, 2015. 2

[7] L. McInnes et al. Umap: Uniform manifold approximation and projection for dimension reduction. (arXiv:1802.03426), September 2020. arXiv:1802.03426 [cs, stat]. 4

[8] K. He et al. Deep residual learning for image recognition. (arXiv:1512.03385), December 2015. arXiv:1512.03385 [cs]. 4

[9] A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. (arXiv:2010.11929), June 2021. arXiv:2010.11929 [cs]. 4

[10] G. I. Parisi et al. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019.

[11] C. Zhuang et al. Unsupervised neural network models of the ventral visual stream. *Proceedings of the National Academy of Sciences*, 118(3):e2014196118, January 2021.