Lightweight Concept Learning for Generalized Visual Reasoning

Joshua J. Daymude Raymond G. Nucuta Andrew C. Smith

Systems Imagination, Summer Internship 2019

Introduction

Visual understanding and reasoning are cornerstones of daily life. As humans, we intuit others' emotions by their facial expressions, drive safely by observing our surroundings, and make our livelihoods by interacting visually and spatially with our world. To achieve a future where intelligent tools help us accomplish these tasks (e.g., self-driving vehicles or automated factories), computer vision must be able to identify all relevant objects in its field of view, understand how these objects are related to one another, and reason about the implications of these relationships. A fire in a fireplace or fire pit is perfectly normal; a building or tree on fire should be interpreted as dangerous and reported to the fire department. A dog being walked on a leash is normal; a dog with a collar running outside of a park might be lost.

In this work, we investigate two approaches to *visual concept learning*, or learning to identify and compose attributes of objects and relationships between them instead of strictly recognizing the objects themselves. The first, called the *Neuro-Symbolic Concept Learner (NS-CL)* [Mao et al. 2019], identifies the objects in a scene as well as their attributes and relationships in order to answer natural language questions such as "How many yellow cars are on the right side of the street?" The second, dubbed the *Zero-Shot Conceptual Learner (ZS-CL)*, is an original undertaking by our team. It utilizes *zero-shot learning* (see [Xian et al. 2017] and the references therein) to achieve visual concept learning with significantly less training data and low power consumption during inference. Moreover, when encountering objects or attributes not included in its training set, ZS-CL can make reasonably accurate informed guesses while other models perform poorly.

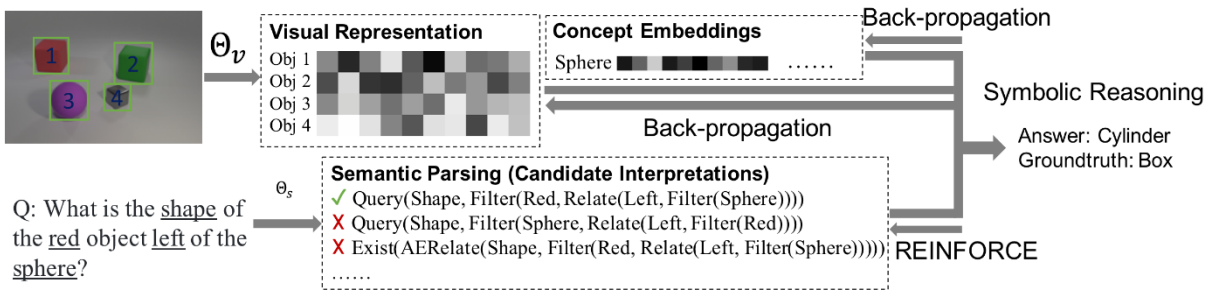
NS-CL

The Neuro-Symbolic Concept Learner (NS-CL) was introduced in [Mao et al. 2019]. It proposed a novel way of performing visual concept learning in a naturally supervised manner, using *visual question answering (VQA)* [Gan et al. 2017] as its metric for success. Informally, its optimization problem can be stated as: "given a scene (image) and a natural language question about the scene, maximize the probability of answering the question correctly." This deceptively simple objective requires many parts of NS-CL to work in tandem. Only after correctly (a) identifying the objects in the scene and extracting their visual concepts/relationships and (b) understanding what the natural language question is asking can NS-CL hope to obtain a correct answer. Incorrectly answering a question may have been the fault of not identifying an object, identifying an object that doesn't exist, incorrectly extracting an attribute, misunderstanding the question, or a combination of any of these.

NS-CL boasts several advantages over its contemporaries. First, it *does not require image annotations* to learn object concepts and relationships between objects. Second, it requires *significantly less training* before its performance matches that of prior works. Finally, because it learns object concepts instead of actual objects, it can *learn additional visual concepts and compositions with little training* (for example, after learning seven colors by training on 1000 objects of each color, NS-CL could learn an eighth color with as few as 100 object instances).

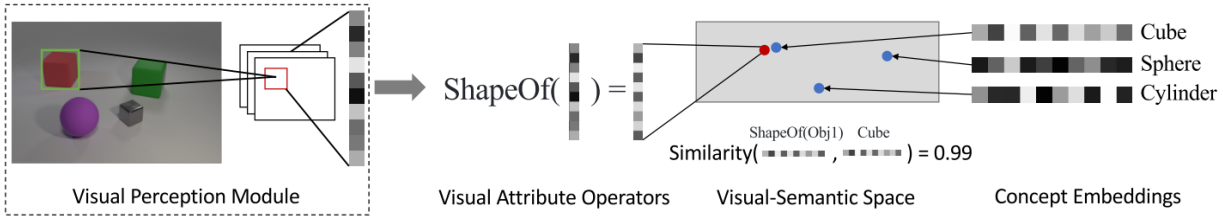
Architecture & Implementation

The NS-CL pipeline contains three major components: a *visual perception module* that extracts object information from a scene, a *semantic parser* that produces an executable program from a natural language question, and a *program executor* that runs the program given by the semantic parser and obtains an answer. A high-level diagram of NS-CL is shown below.



Visual Perception

The visual perception module takes an image as input and ultimately outputs a set of concept and relational embeddings for each object in the scene. As a pre-processing step, each image is annotated with bounding boxes around each object by a pretrained object detector (Mask R-CNN). A ResNet-34 model is then used to extract region-based features of each detected object and image-based features of the entire scene. A set of attribute operators (one for each concept, such as color, size, etc.) embed an object's features into a specific concept space (e.g., color space, size space, etc.), where it can be compared to embeddings of known attributes (e.g., blue, green, yellow, etc.). The closer the object's concept embedding is to a known attribute, the more certain the model can be that the object has that attribute. A similar process is used to understand relationships between objects, using a combination of the object and scene features. Thus, NS-CL ultimately has conceptual and relational understanding of each of the objects in the scene.



Semantic Parser

The semantic parser "compiles" the natural language question about the scene into an executable program. These programs represent the semantics of a natural language question as a hierarchy of nested operations taking attributes and concepts as inputs. These operators and the attribute/concept vocabulary are defined in a *domain-specific language (DSL)*. The DSL changes depending on the dataset; for example, when parsing questions about the CLEVR dataset of simple geometric objects, the DSL includes concepts like "material" and attributes like "metal" or "matte". In order to properly parse "how many cubes are behind the red matte sphere?" the DSL needs operators for filtering out objects without certain concepts (e.g., "red" or "matte") and those without certain relationships (e.g., "behind"). The semantic parser's recursive structure also allows it to process questions of varying length and complexity with ease.

Program Executor

NS-CL combines the conceptual and relational features extracted during visual perception with the parsed semantic program representing the question in the program executor. The program executor

deterministically runs the semantic program using the visual features as input. At each operator, the executor computes an attention mask over all objects in the scene. Each element of the mask represents the probability that the object corresponding to that element fits the description of the set. For example, if the operator wants to filter out all red objects in the scene and only object 4 is red, the mask may be {0.01, 0.02, 0.01, 0.99}. This process continues until all masks for the operators of the semantic program have been completed, and a final answer is then computed and returned.

Implementation

An open source implementation of NS-CL is available on GitHub [Mao et al. 2019] and is heavily dependent on Jacinle [Mao 2019], a personal Python toolbox written by one of the NS-CL authors. As it stands, this version of NS-CL is very brittle, non-modular, and specific to the dataset that the NS-CL paper performs most of its experiments on. Moreover, the code structure does not map directly onto NS-CL's description in the paper. Our version of NS-CL, which is in the SystemsImagination GitHub, largely focuses on three goals: (a) improving NS-CL's code modularity and quality, (b) extending NS-CL's capabilities, and (c) applying NS-CL to new application domains beyond the limited examples in the paper. The reasoning and design behind many of our changes are carefully documented in our Teams Wiki. Unfortunately, as we will explain in **Discussion**, we were unable to achieve these goals due to NS-CL's poor code quality.

Experiments & Results

Before improving NS-CL and extending its capabilities and applications, we set out to validate the base implementation's performance by replicating the experiments in the paper. We identified 12 possible experiments and carefully documented their data requirements, procedure, and reported performance in the NS-CL paper. However, many of these experiments were practically impossible to replicate. Some required datasets used by the NS-CL authors that were not publicly available in their cleaned form. Others required pretrained models that the authors neither referenced nor provided. A few even required NS-CL to perform parameter reloading or report custom metrics, neither of which are features the base implementation supports. In addition to all of this, quite a few experiments weren't fully documented in the NS-CL paper, which left us guessing various hyperparameters (which we documented in detail in our Wiki). These obstacles proved stubborn, even after contacting the NS-CL authors multiple times for an explanation of the experiments they performed. We address this further in **Discussion**.

We did, however, complete one important experiment and obtained reasonable results. The goal of this experiment (dubbed **Experiment 4** in our Teams Wiki) was to evaluate how NS-CL's accuracy varies by type of query a natural language question represents. For example, we compared NS-CL's performance on counting ("how many...?") queries to that for existential ("is there a...?") queries. In keeping with the parameters outlined in the paper, we trained NS-CL on 5,000 images from the standard CLEVR dataset with 20 question/answer pairs per image. We then validated on another 5,000 images and 20 question/answer pairs, resulting in the following.

Query Type	Overall	Count	Compare Number	Exist	Attribute	Compare Attribute
NS-CL Paper	98.9%	98.2%	99.0%	98.8%	99.3%	99.6%
Ours	98.5%	97.5%	98.9%	99.1%	98.9%	98.7%

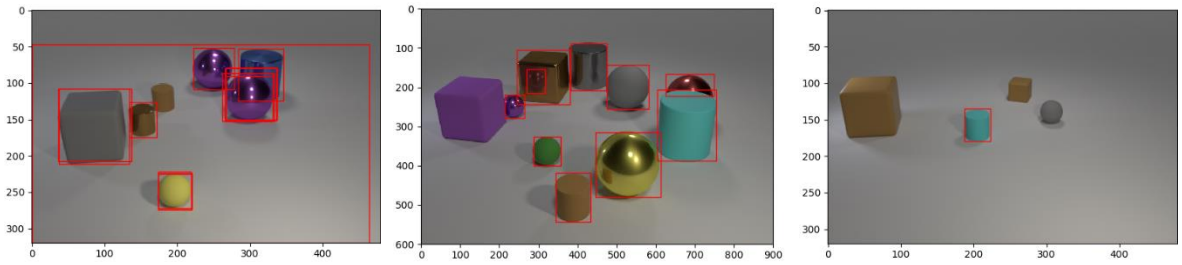
These results are relatively consistent with the paper's claims. However, this was the only experiment that used data the authors made available and which required no modifications to NS-CL's code, so close alignment between our experiments and the stated performance was expected.

Discussion

Several aspects of NS-CL led to the eventual decision to abandon it for something we had more control over. As a piece of software, NS-CL is neither modular nor self-contained. Its main author, Jiayuan Mao, relies heavily on her own Python library called Jacinle [Mao 2019] that makes very specific and arbitrary changes to standard libraries (such as PyTorch). This made NS-CL's implementation brittle in ways that were difficult to debug; we often struggled to compare her forked versions of PyTorch and other common utilities with the standard ones, and even found several absolute paths specific to Mao's hardware that caused crashes. All of this was exacerbated by a complete lack of documentation throughout the code and meaningful differences between the implementation and NS-CL's description in the paper. These issues seem to indicate that NS-CL was put out on GitHub more for the researchers to say there was an implementation available rather than to provide a general, flexible platform for others to experiment and improve upon.

Poor software quality aside, NS-CL is significantly under-featured compared to what was described in the paper. Many interesting datasets featured in the paper — such as the Minecraft dataset [Wu et al. 2017, Yi et al. 2018] and VQS dataset [Gan et al. 2017] — are not supported by the NS-CL implementation. Even the CLEVR dataset [Johnson et al. 2017], which is the only one supported by the base implementation of NS-CL, requires nontrivial reformatting that is not documented by the authors

(we've since documented it explicitly in our Teams Wiki). But the most problematic missing feature is an object detection pipeline, responsible for putting bounding boxes on each object. This critical first step of the NS-CL framework is handwaved as "pre-processing" by the authors. Attempting to replicate their efforts, we used state-of-the-art pretrained Mask R-CNN models (available via Facebook's R-CNN Benchmark [Massa et al. 2018]) for object detection. However, while these models were decent at detecting real-world images of streets, animals, etc., they were quite poor at finding objects in CLEVR images. Mask R-CNN would often detect the same object many times while missing others or detect a significant fraction of the entire scene as one object (see below, left). After filtering these bad bounding boxes, it became clear that while it was relatively successful for some images (e.g., below, center), it completely missed far too many objects in others to be useful in visual reasoning (e.g., below, right).



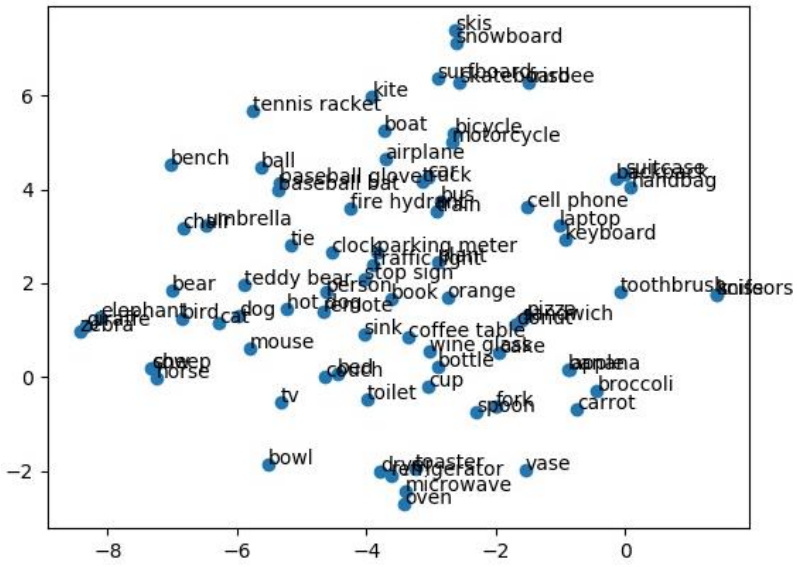
ZS-CL

Moving into the next phase of our project, we worked to take the best of NS-CL forward while leaving the cumbersome code and missing data behind. This resulted in original work on a model known as the *Zero-Shot Conceptual Learner (ZS-CL)*. With ZS-CL, we aimed to design a machine learning platform for visual reasoning that (a) learns and utilizes generalizable concepts, (b) performs reasonably when encountering new/unrecognized data, (c) uses significantly less training data than prior works, (d) consumes relatively low power during inference, and (e) allows for online or iterative learning. We envision this approach being useful for autonomous event detection and response in drones, whose battery life, storage capacity, and computational capabilities are all limiting factors.

Background

ZS-CL combines conceptual learning (previously discussed in **NS-CL**) with *zero-shot learning* (ZSL) [Xian et al. 2017], an indirect approach to classification that allows models to classify items with reasonable accuracy even when no instances of such items were included in the training set. Essentially, ZSL enables models to make "educated guesses" about unrecognized items based on their similarity to known items. Since ZSL is a less known form of machine learning, we give a brief background here.

In ZSL, all class labels are first embedded into N-dimensional word space; in our case, we use Word2Vec [Mikolov et al. 2013 ICLR, HLT, NIPS] where N=300. In word space, labels representing similar ideas are grouped closer together (see, e.g., below). The set of all classes is then partitioned into *training classes* and *zero-shot classes*. A classifier is trained on instances of the training classes only, with the important property that the second-to-last layer has N nodes, the last layer has C nodes (where C is the total number of classes), and the weights between these layers are fixed, representing the word space embeddings of the class labels. Effectively, this means the classifier is being trained to embed visual features into word space close to where the label of the features' class is. Then, when performing inference, the final layer of fixed weights is removed. The model thus embeds visual features into word space, and the k-closest labels in word space are used as the top-k classifications.

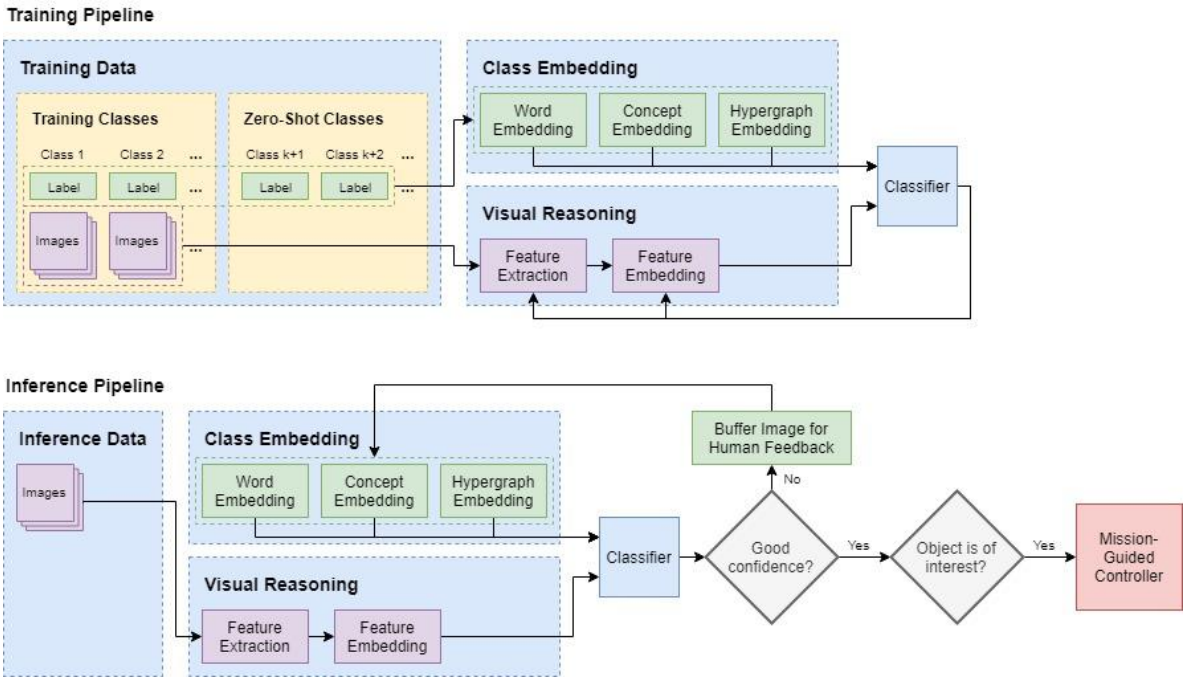


Since zero-shot learning allows us to train on only a fraction of the classes while still achieving reasonable accuracy on the zero-shot (unknown) classes, it is well aligned with our goals of training lightweight object recognition models with little data. Moreover, since the classifiers can be relatively simple, it ensures the number of weights – and thus the memory footprint of the model – remain small.

Architecture & Implementation

The training and inference pipelines for ZS-CL are shown below. As described in **Background**, ZSL splits the training data into training and zero-shot classes. All class labels (training and zero-shot) are embedded into word space; we also envision embedding objects' concepts and hierarchical information. Each input image is assumed to be cropped to an object's bounding box. These object images are then fed through pretrained feature extractors (such as ResNet*, VGG*, or Xception) and are embedded into

word/concept/hierarchical space by feature embedders. Finally, a classifier can simply look for the closest neighbors in embedding space to perform classification. During inference, ZS-CL can assess its confidence in its classification to either act according to its deterministic, mission-guided controller (if its confidence is high) or buffer the image and its guess for later feedback (if its confidence is low).



As of this writing, much of this work is not yet complete. Our implementation of ZS-CL is available on GitHub [Daymude et al. 2019]. It currently provides a fully modular pipeline that includes dataset preparation (using the VisualGenome dataset [Krishna et al. 2016]), word embeddings, feature extraction, feature embedding, and training/evaluation scripts. Notably, this has not yet addressed conceptual or hierarchical information, and does not yet support any kind of online/iterative learning.

Experiments & Discussion

Our preliminary experiments with ZS-CL focus on varying different hyperparameters in order to understand the viability of this approach/implementation for our goals. In particular, we are interested in the impact of (a) the feature extractor used, (b) the structure of the feature embedder model, (c) the training/zero-shot class partition, and (d) the dimensionality of the word embeddings on ZS-CL's validation and zero-shot classification accuracies.

The two datasets we use are both subsets of the VisualGenome dataset [Krishna et al. 2016]. The first, dubbed COCO, uses the 80 classes from the standard COCO dataset [Lin et al. 2014] and

samples 500 instances of each from the VisualGenome corpus. The second, dubbed VisGen750, uses 447 classes for which there are at least 750 instances in the VisualGenome corpus, combining singular and plural versions of class labels. Both are generated using our imgs2objs script; the details of this data collection can be found in our Teams Wiki.

Comparing Feature Extractors

Our first experiment analyzed the impact of the feature extractor used. Although each extractor's performance is already well-known in isolation, it was important for us to establish how they interact with ZSL. Using our COCO dataset, we extracted features with VGG16, VGG19, MobileNetV2, ResNet50, and Xception. Then, for each extractor, we trained the feature embedder on 75% of the classes for 50 epochs, using a 95% training/validation data split and a batch size of 128. The following table shows the top-{1, 3, 5} categorical accuracy achieved on the training classes (validation data) and zero-shot classes.

Model	**VGG16**	**VGG19**	**MobileNetV2**	**ResNet50**	**Xception**
Top-1 training	53.61%	54.69%	46.03%	61.90%	**66.09%**
Top-3 training	71.79%	73.45%	67.68%	78.57%	**82.68%**
Top-5 training	80.38%	80.74%	76.70%	84.27%	**87.09%**
Top-1 zero shot	4.05%	1.39%	1.98%	4.61%	**5.94%**
Top-3 zero shot	12.10%	11.80%	8.84%	14.33%	**15.54%**
Top-5 zero shot	17.99%	20.69%	18.09%	19.55%	**23.55%**

When evaluating these results, we consider a tradeoff between model accuracy and memory footprint. Xception is the clear frontrunner in terms of accuracy, outperforming all other models on both the training and zero-shot classes. At 88MB, it is also a sixth of the size of the VGG models, and roughly the same size as ResNet50. MobileNetV2, on the other hand, is only 14MB but achieves the worst accuracy among the five we evaluated. Since one of our goals is to be as lightweight as possible, either of these could be appropriate choices in the long run, though how much accuracy we would sacrifice in return for minimal memory footprint remains an open question.

Comparing Feature Embedder Architectures

We next experimented with slight variations on the architecture of the feature embedder. The base architecture is a simple sequential network that reduces layer sizes by powers of 2 and has dropout on some of the bigger layers. Using the features extracted from the COCO dataset with Xception using the same training/zero-shot class split as above, we evaluated the following architectures:

- **C1:** 1024 -> Drop(0.8) -> 512 -> Drop(0.5) -> 256
- **C2:** 1024 -> Drop(0.8) -> 512 -> Drop(0.8) -> 256
- **C3:** 2048 -> Drop(0.8) -> 1024 -> Drop(0.5) -> 512 -> Drop(0.5) -> 256
- **C4:** 2048 -> Drop(0.8) -> 1024 -> Drop(0.5) -> 512
- **C5:** 1024 -> Drop(0.8) -> 512 -> Drop(0.5) -> 512 -> Drop(0.8) -> 256
- **C6:** 1024 -> Drop(0.7) -> 512 -> Drop(0.7) -> 256
- **C7:** 1024 -> Drop(0.8) -> 512 -> Drop(0.6) -> 300

Configuration	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
Top-1 training	66.09%	62.27%	65.44%	67.10%	57.43%	**67.53%**	65.51%
Top-3 training	82.68%	79.58%	81.96%	**83.84%**	76.26%	82.47%	81.46%
Top-5 training	87.09%	85.21%	86.72%	**89.68%**	83.04%	88.31%	86.44%
Top-1 zero shot	**5.94%**	4.61%	5.38%	4.09%	3.83%	5.75%	2.77%
Top-3 zero shot	**15.54%**	13.03%	14.92%	12.71%	10.14%	15.12%	10.60%
Top-5 zero shot	23.55%	18.25%	**24.18%**	21.89%	14.73%	21.96%	20.31%

As shown in the table, the base network (**C1**) performs well on both training and zero-shot classes, while most of the variations only diminish performance. Configuration **C4** does achieve slightly better training class accuracy, but lacks good zero-shot performance and is a noticeably denser neural network. Going off this data, we decided to stick with our original configuration, though more detailed sweep will provide comprehensive results. A better understanding of the ZSL and modern classification literature could also yield higher performant architectures for ZS-CL.

Comparing Data Partitions

Next, we investigated the impact of the training/zero-shot class split on ZS-CL's performance. There are some subtle effects at play with ZSL. On one hand, increasing the proportion of training classes gives ZS-CL more data to learn from. However, the literature suggests that leaning too heavily on training classes seems to deter ZSL models from classifying items as any zero-shot class during inference, causing poor zero-shot accuracy. On the other hand, not giving a ZSL model enough training classes to learn from is also an obvious barrier to good performance. In a sense, these dual issues present equally problematic opportunities to “overfit,” so finding a reasonable training/zero-shot class split is important. To characterize these effects, we trained the feature embedder on different splits of the COCO dataset’s 80 classes, using the features extracted with Xception. The remaining hyperparameters were the same as in the previous feature extractor experiment.

#Training/#Zero-Shot	40/40	50/30	58/22	70/10
Top-1 training	**70.13%**	68.68%	66.09%	65.57%
Top-3 training	**84.30%**	**84.30%**	82.68%	79.98%
Top-5 training	89.50%	**89.67%**	87.09%	85.59%
Top-1 zero-shot	5.54%	5.20%	5.94%	**8.12%**
Top-3 zero-shot	15.88%	16.86%	15.54%	**16.87%**
Top-5 zero-shot	25.31%	**26.00%**	23.55%	21.52%

It's difficult to draw meaningful conclusions from this data, other than that it should be rerun on a larger dataset with a more fine-grained sweep of splits. Across the different splits, the training accuracies are all relatively similar (within 5%), generally performing better when there are less training classes to learn. The spread is even tighter for the zero-shot accuracies (largely within 3%), though the 70/10 model achieves significantly better top-1 accuracy than the rest.

However, the size of the splits only captures part of this effect; it is also important to understand which classes are being used in each split. The COCO classes can be broadly grouped into categories like "vehicles", "animals", "household items", etc. Intuitively, treating all vehicles as zero-shot classes will produce different behavior than splitting the vehicle classes evenly in the partition. In the former, the model will have no prior exposure to vehicles with which to "compare" a new object; in the latter, the model's training on "trucks" would help it classify a "car" because of their similarity. All of the above experiments were performed by manually splitting the classes so both the training and zero-shot classes had representatives from each category. To perform a preliminary investigation into the effect of categorical learning, we instead experimented with restricting the set of training/zero-shot classes to an individual category, "household items". The results are as follows.

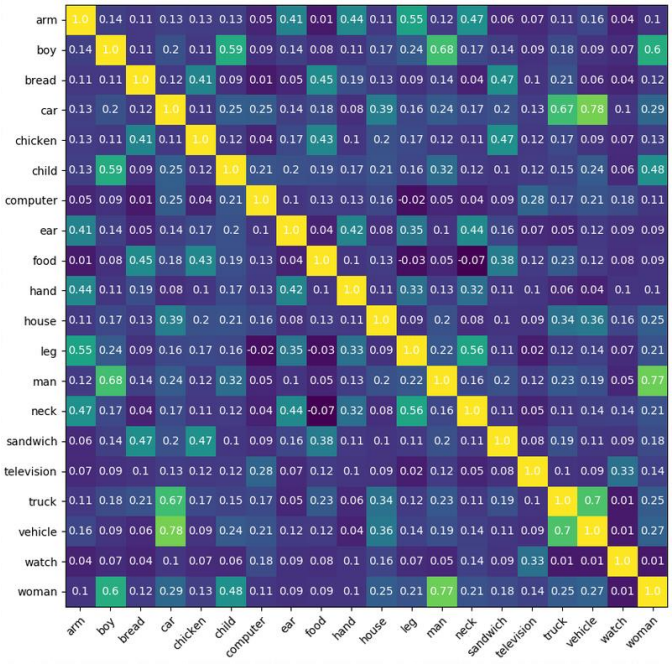
#Training/#Zero-Shot	35/15
Top-1 training	70.76%
Top-3 training	84.76%
Top-5 training	90.16%
Top-1 zero-shot	0.28%
Top-3 zero-shot	5.20%
Top-5 zero-shot	18.20%

The training accuracies for this category-specific model are better than those for the general models above while the opposite trend holds true for the zero-shot accuracies. This is likely because ZSL

models need both similarity and diversity in their training data in order to “cluster” concepts and object types together in its understanding. While many household objects may be related in the eyes of a human, a human’s universe of knowledge extends far beyond this category. These category-specific models have nothing to distinguish between, so even similar objects appear unrelated. This is a problem we think can be addressed by avoiding category-specific datasets, emphasizing broad diversity in images and object classes.

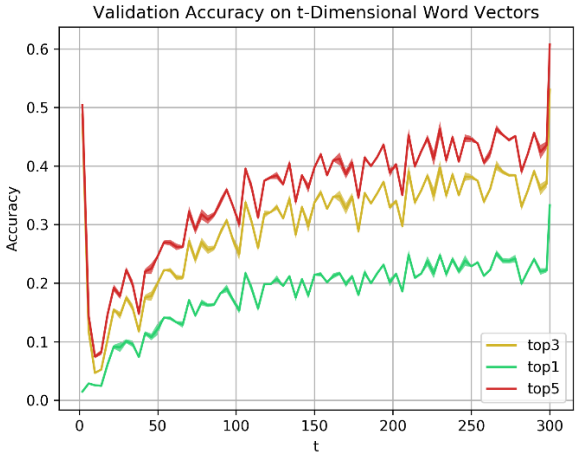
Reducing the Dimensionality of the Word Embeddings

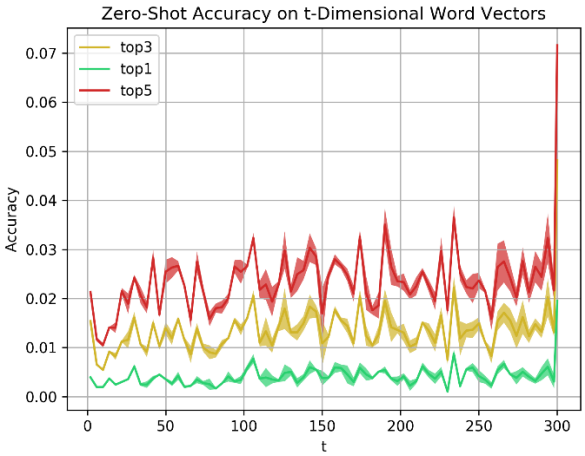
As a final experiment, we investigated how word space dimensionality impacts ZS-CL's performance. The Word2Vec word vectors are in 300-dimensional space, and largely cluster similar things together while dispersing dissimilar categories (see the heatmap below showing cosine similarity between the classes of the COCO dataset). Using *t-distributed stochastic neighbor embedding* (t-SNE) [van der Maaten 2008], a popular dimension-reduction algorithm, we experimented with reducing the dimensionality of the word vectors and then evaluating validation and zero-shot accuracy. If t-SNE could maintain the similarity/diversity structure present in 300-dimensional space but in lower dimensions, the memory footprint of our approach would be even smaller without sacrificing accuracy.



Using an experimental framework optimized for multi-GPU support, we performed three independent trials for each value of t in {2, 6, 10, …, 298}, training the feature embedder on features extracted from the VisGen750 dataset using Xception. We used a 50% training/50% zero-shot class split

and trained each feature embedder for 50 epochs. The following two graphs show validation and zero-shot accuracy, respectively, as a function of word space dimension. Note that the right-most point represents training with the original 300-dimensional word vectors, where t-SNE was not applied.





These results are surprising for a few reasons. First, while dimension-reduction generally hurts the model's accuracy significantly — top-1 training accuracy dives from roughly 34% to 22% just by reducing the dimension from 300 to 298 — the model performs significantly better in 2-dimensional word space than most other lower dimensions. Second, the general accuracy trends between the validation and zero-shot classes are very different: validation accuracy generally decreases as the dimension is reduced while zero-shot accuracy largely remains the same. Finally, the accuracy trends are far from smooth: jagged peaks and valleys suggest a sensitivity to not only the dimensionality of the word vectors, but also the way t-SNE arranges the words themselves.

Conclusion & Future Work

There is much left to improve and understand when it comes to ZS-CL, especially in bringing it closer to our vision of a lightweight, concept-driven learning model. While we've established modular and flexible functionality, we still aren't doing conceptual, hierarchical learning. While we've pushed into several approaches to minimize training data needs (namely, zero-shot learning and dimension reduction), these adaptations have come at the cost of accuracy. And while we envision deploying ZS-CL on swarm platforms like autonomous drones, we have yet considered what design and implementation considerations are needed to make that a reality. These are all grounds for future work.

References

[Daymude et al. 2019] Joshua J. Daymude, Raymond Nucuta, and Andrew Smith. ZS-CL: The Zero-Shot Conceptual Learner. Accessed July 26, 2019. 2019. Implementation available online at https://github.com/SystemsImagination/ZS-CL.

[Gan et al. 2017] Chuang Gan, Yandong Li, Haoxiang Li, Chen Sun, and Boqing Gong. VQS: Linking Segmentations to Questions and Answers for Supervised Attention in VQA and Question-Focused Semantic Segmentation. In the *IEEE International Conference on Computer Vision* (ICCV 2017). 2017.

[Johnson et al. 2017] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *International Conference on Learning Representations* (ICLR 2017). 2017. Data available online at https://cs.stanford.edu/people/jcjohns/clevr/.

[Krishna et al. 2016] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. On arXiv at https://arxiv.org/abs/1602.07332.

[Lin et al. 2014] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*. Dataset available online at http://cocodataset.org/#home.

[Mao et al. 2019] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences from Natural Supervision. In *International Conference on Learning Representations* (ICLR 2019). 2019. Implementation available online at https://github.com/vacancy/NSCL-PyTorch-Release.

[Mao 2019] Jiayuan Mao. Jacinle. Accessed May 22, 2019. 2019. Implementation available online at https://github.com/vacancy/Jacinle.

[Massa et al. 2018] Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. Accessed June 14, 2019. 2018. Implementation available online at https://github.com/facebookresearch/maskrcnn-benchmark.

[Mikolov et al. 2013 ICLR] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations* (ICLR 2013). 2013.

[Mikolov et al. 2013 HLT] Tomas Mikolov, Wen-tau Yih, and Geoffery Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (NAACL HLT 2013). 2013.

[Mikolov et al. 2013 NIPS] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Neural Information Processing Systems* (NIPS 2013). 2013.

[van der Maaten 2008] L. J. P. van der Maaten and G. E. Hinton. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9:2517-2605, 2008.

[Wu et al. 2017] Jiajun Wu, Joshua B. Tenenbaum, and Pushmeet Kohli. Neural Scene De-Rendering. In *International Conference on Learning Representations* (ICLR 2017). 2017.

[Xian et al. 2017] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-Shot Learning – The Good, the Bad, and the Ugly. In *2017 IEEE Conference on Computer Vision and Pattern Recognition* (CVPR 2017). 2017.

[Yi et al. 2018] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Neural Information Processing Systems* (NIPS 2018). 2018.