CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 70 **October 31, 2017**

Object-Oriented Deep Learning

by

Qianli Liao and Tomaso Poggio
Center for Brains, Minds, and Machines, McGovern Institute for Brain Research,
Massachusetts Institute of Technology, Cambridge, MA, 02139.

Abstract:

We investigate an unconventional direction of research that aims at converting neural networks, a class of distributed, connectionist, sub-symbolic models into a symbolic level with the ultimate goal of achieving AI interpretability and safety. To that end, we propose Object-Oriented Deep Learning, a novel computational paradigm of deep learning that adopts interpretable "objects/symbols" as a basic representational atom instead of N-dimensional tensors (as in traditional "feature-oriented" deep learning). For visual processing, each "object/symbol" can explicitly package common properties of visual objects like its position, pose, scale, probability of being an object, pointers to parts, etc., providing a full spectrum of interpretable visual knowledge throughout all layers. It achieves a form of "symbolic disentanglement", offering one solution to the important problem of disentangled representations and invariance. Basic computations of the network include predicting high-level objects and their properties from low-level objects and binding/aggregating relevant objects together. These computations operate at a more fundamental level than convolutions, capturing convolution as a special case while being significantly more general than it. All operations are executed in an input-driven fashion, thus sparsity and dynamic computation per sample are naturally supported, complementing recent popular ideas of dynamic networks and may enable new types of hardware accelerations. We experimentally show on CIFAR-10 that it can perform flexible visual processing, rivaling the performance of ConvNet, but without using any convolution. Furthermore, it can generalize to novel rotations of images that it was not trained for.

This work was supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF - 1231216.

Contents

1 Introduction

We propose a novel computation paradigm of deep learning based on the concept of "objects", in contrast to the traditional "feature-oriented" deep learning. It can potentially lead to better efficiency, sparsity, flexibility, transformation disentanglement/invariance, background tolerance, low-level parallelism, new options of hardware accelerations, and most importantly, more interpretability. The main motivations behind our proposal are as follows:

1.1 Towards Symbols

Traditionally, (deep) neural networks are distributed, connectionist, sub-symbolic models. However, human high-level thoughts appear to be mostly symbolic, centralizing on objects and their relations. It is a question where in the sensory perception pathways, sub-symbolic signals start to give rise to symbols.

Motivated by the above question and one of CBMM's research thrusts "Towards Symbols" (CBMM NSF proposal, January 2017), we would like to see to what extent we can convert neural networks, sub-symbolic models, into a symbolic level. To that end, we will attempt to assign symbols "first-class citizen" status in our new form of neural networks. In traditional deep learning, almost all intermediate representations are **N-dimensional tensors of distributed features**. There is no explicit representations of symbolic concepts like objects, poses, positions, part-whole belongings, objectness, velocities, events, etc. We call this traditional form of deep learning "feature-oriented" deep learning (FODL).

In our framework, we aim at making progress on explicitly representing all such symbolic concepts, leading to what we call "object-oriented" deep learning (OODL). In this framework, the basic representational atom is called an "object" (in contrast to N-dimensional tensors in FODL). The concept "object" generally refers to a discrete symbol containing some properties, similar to that of object-oriented programming [1] As a foundamental difference from FODL, we require all intermediate representations in our network to be (1-D) **lists of "objects/symbols" (and their associated properties)** as shown in Figure 1. We will show in this report that we can use 1-D lists of objects to perform flexible visual processing, rivaling the performance of a ConvNet, but without using any convolution.

1.2 Making Neural Networks Not "Neural"? One Alternative to AI Safety & Interpretability

Recently there is a flurry of exciting work focusing on interpretability and AI safety, most of which try to draw some interpretation from the distributed code of learnt deep neural networks.

Here we provide another (somewhat radical) alternative to this issue: What if we drop (as much as possible) distributed coding? What if we eliminate as much as possible non-symbolic components from neural networks? What if we make neural networks not "neural"?

That said, we are not claiming it is absolutely possible to make neural networks completely symbolic while maintaining its excellent performance. But rather, we claim that it is a direction worth of pursuing, if one wants to have more interpretability in such systems, as an alternative to fighting with distributed code. In this work, we make one of the first steps in this direction.

In contrast to more conventional views that we should only build high-level symbols on neural infrastructure and "ground symbols" on neural representations, we note that there could be more extreme approaches: would it be possible to "erode" the foundation of neural networks by symbolizing its most basic computations? Would a fully-symbolic network be preferred due to its enhanced interpretability? These are controversial questions, and we invite the readers to add more thoughts to it.

[1] The fact that people can use object-oriented programming to represent almost everything is an encouraging indication that this basic representational atom would be versatile enough for a wide range of tasks.

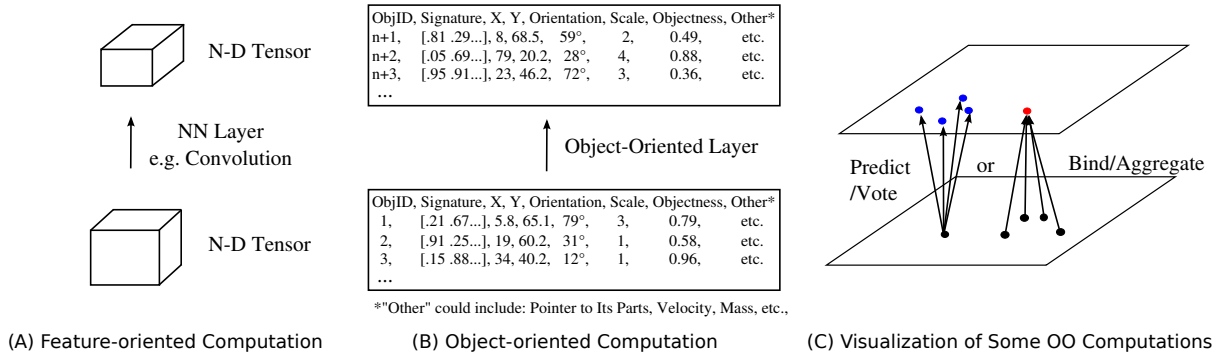


Figure 1: A comparison between Feature-Oriented and Object-Oriented computations. Object-Oriented computations use an "object" as its representational atom, an entity where useful, **disentangled, interpretable properties can be packaged**. Such properties are maintained throughout all layers, aiming at providing an interpretable and complete characterization of visual perception. For visual processing, the coordinates X and Y, representing the centers of discovered objects, are global in the original image coordinate system throughout all layers, which is more intuitive than ConvNet's implicit layer-local coordinate systems. "Pointer to parts" entry records which objects (and by what amount) contributes to this object, which is good for segmentation purposes. Objectness represents the probability that the entry is an object, which is useful when pruning is needed (to achieve faster processing). Unlike the fixed computational cost of feature-oriented computations, the cost of any object-oriented computation is proportional to the number of input objects, which can vary from sample to sample.

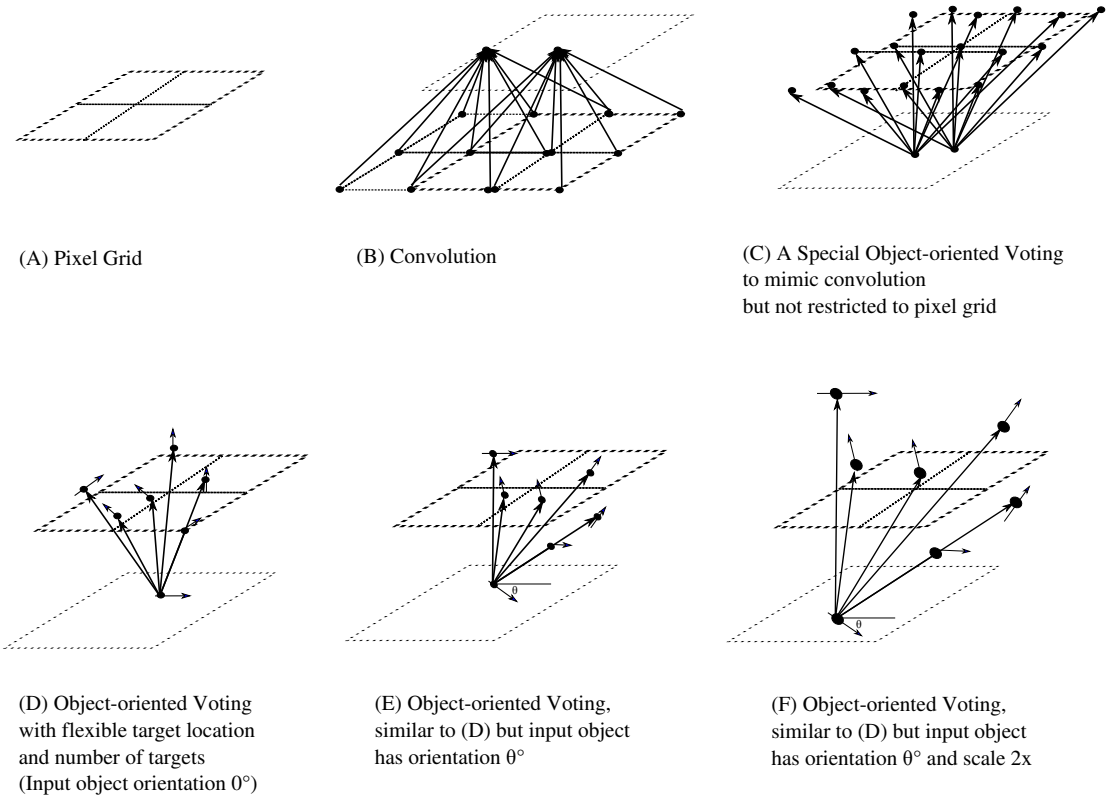


Figure 2: We compare Convolution and Object-Oriented Voting (a more general replacement of convolution). Note that one significant difference from convolution is that from (C) to (F) the positions of voting and voted objects are in the global image coordinate system throughout all layers.

1.3 Object-Oriented vs. Feature-Oriented Processing: Towards a Representation of Objects in Visual Processing

Although humans intuitively understand/interpret the world in terms of discrete objects. State-of-the-art machine vision systems do not have a representation of visual objects — they are "feature-oriented" in the sense that it models the world as a continuous, statistical blob of features with little (if not none at all) **grouping of signals**. This representation severely limits what we can do in a wide variety of visual processing tasks. For instances, following issues clearly require more advanced representations beyond a blob of ungrouped features:

1. "Binding problem": after recognizing multiple properties of the visual content (e.g., rotation, material, color, etc.), one need to associate them to one entity. Suppose there are two objects placed relatively close next to each other: a red rectangle tilted at 30 degree and green triangle at 60 degree. How does a visual system know that the color red and angle 30 are associated with rectangle and color green and angle 60 are associated to the triangle? Without explicit binding, one can only train some ad-hoc classifier on the blob of features of ConvNet to obtain some guess. With our object-oriented framework, all the properties are intrinsically associated to objects. When an object is predicted, its properties are predicted as well.
2. Recognition or fine-grained visual analyses under occlusion requires filtering out occluders. One has to know which features belong to which object (e.g., foreground vs. background), possibly in a very fine-grained manner.
3. In a vision systems that is agnostic to objects, when an object is recognized, it is unclear whether it is recognized by itself or by context (i.e., periferal content within the same rectangular receptive field, as in ConvNet) because there is technically no difference between an object and background in such statistical learning machines. Such machines may have trouble recognizing a novel object or an object on a novel background unless one train it with almost infinite amount of data to cover all situations.
4. Visual (scene) imagination requires compositional rendering of a number of objects on a "canvas", possibly in a partially overlapping way. Each object should be rendered independently, and changing the pose of one object should not affect the background, even if they are very close retinotopically (e.g., seeing an animal through a fence in a zoo). A feature-oriented generator (e.g., a deconvolution net) renders a block of features at a time, where features of different objects are not distinguished and will clearly interfere each other.

Due to above reasons, we argue that a better visual system should have some internal **feature binding**, **property association** and **signal grouping**. This requires representing an entity that features, signals and properties are binded/grouped/associated to — this is what we defined as an "object": It is a discrete container that explicitly packages properties, feature bindings, part-whole belongings, etc. We can treat lowest-level edges or pixels as first level objects and then build more complex objects from bottom up. An object at every layer has explicit knowledge of which lower level objects constitute it. Every object contains common properties like its rotation, size, etc., which will help predicting other objects and their associated properties.

1.4 Disentangled Representations and Invariance

Disentangled representations is one of the "holy grails" of deep learning. How to learn disentangled representations from data using distributed code is however a challenging task. Variational Auto-encoder [1] can do it to some extent, but cannot so far scale to real-world tasks. We call these approaches "neural disentanglement".

In our object-oriented deep learning, we take a "symbolic disentanglement" approach — we tackle the problem in an explicit way by making disentangled factors as symbolic properties of an object. When an object is detected, its properties (e.g., pose, size, position) are predicted at the same time. Same things happen in every intermediate layer of the network, so that one has full knowledge of the properties of any detected object and its parts, recursively throughout all layers.

Of course, one limitation is that our approach currently requires some prior knowledge of what disentangled factors (e.g., positions, rotations, scalings, etc.) we want to deal with. But we argue that these are the most important

factors — disentangling them, either symbolically or neurally, is an important first step to solving the problem of disentangled representations.

Also, an "object" in our framework can be extended to contain other properties such as velocity, acceleration, mass, time, events, etc. Incremental but tangible progress can be made to account for all disentangled factors that we care about regarding an object of any domain.

As an example of what we can do with disentangled representations, we show that we can perform invariant object recognition by training on one rotation of objects (i.e., standard canonical view) and generalize to other rotations (360 degrees) without re-training: In our framework, when an object is detected, its rotation is also detected. At the end of the network, rotation invariance is achieved by simply discarding the pose.

1.5 More Flexible and Efficient Visual Processing Compared to ConvNets

So far almost all state-of-the-art machine vision systems rely on convolutions. However, we note that convolution has several potential limitations as follows, which can be addressed/compensated by our Object-Oriented (OO) visual processing:

1. Convolution operates on a **regular rectangular** grid of features. Thus, irregular or sub-pixel/voxel predictions are not naturally available. This seems to be a very artificial limitation. In Object-Oriented visual processing, an object can make arbitrary spatial predictions with floating point sub-pixel precision. For example, in our current implementation, every object predicts next layer objects on a circle around it instead of on a 3x3 square like typical ConvNets.
2. With ConvNets, features resides in regular grid of integer coordinates in some arbitrary coordinate systems, which vary from layer to layer. It does not naturally support high precision localization of objects in a fine-grained manner. In our framework, all objects throughout all layers have continuous positions (x and y) in the **original image coordinate system** — one has an intuitive knowledge of the positions of objects.
3. Convolution, as an atomic operation, aggregates low-level predictions **uniformly (spatially) in an unconditional way**. We argue that it might not be fine-grained enough as an atomic operation for future development of deep learning (and computational neuroscience). In the context of modeling the brain, convolution is **a cell-level operation** — it does not support flexible synapse-wise heterogeneous computations: e.g., multi-compartment synaptic model, feature binding and/or synapse-soma coincidence detection. In object-oriented visual processing, we propose two alternative conceptual stages: voting(predicting) and binding (discussed more in later sections). These two conceptual computations jointly capture convolution as a special case but support considerably more possibilities including modeling synapse-soma coincidence detection, synapse-wise conditional aggregation, varying synapse distances from cell body, varying number of synapses per cell (and/or per sample), etc.
4. In ConvNets (and in general feature-oriented deep learning), it is usually not possible to have multiple objects at the same location. If so, their features are mingled. In OO visual processing, we can have arbitrary number of overlapping objects at arbitrary locations. We believe it has better potential to handle feature binding, whole-part belongings and overlapping objects segmentation.
5. Convolution is a **filter-driven** process, enforcing a filter-centric thinking paradigm. When we imagine what convolution-based models do, we think about filtering input matrices. It limits the scope of algorithms we can design. Furthermore, due to the **filter-driven** nature, the computational cost of convolution-based model is largely determined by the number of filters, instead of the number of inputs. One filter can only accept a fixed number of inputs and fixed input dimension. In object-oriented visual processing, we do the other way around — we think in a **object-driven, or data-driven** way, we think about **what each object can predict, can generate, and how can it influence the next layer**. We argue that this might be a more natural way of thinking about vision. For example, detecting a rotated object is simple: two rotated parts naturally predicts a rotated whole at a particular location. If the prediction agrees (as checked by the binding process), the whole is detected as well as its orientation.
6. Sometimes people also say ConvNets are data-driven, but this comment refers to more at a computational graph level. One can eliminate branches if they are not used on a sample by sample manner (if one has a

good dynamic computation graph engine, say DyNet [2] or TensorFlow Fold [3]). But in general it is not common, not very efficient and argueably too coarse-grained. In contrast, our object-oriented deep learning naturally supports **input-driven, heterogeneous, dynamic computations tailored for each sample, within each layer,** in addition to dynamic computation between layers or from batch to batch. This offers a good complement to the recent popular concept of dynamic networks (as in DyNet).

7. Current computer vision systems spend almost the same amount of time to process an empty input (e.g., a piece of white paper) and a clutterred input (e.g., Time Square of New York City), because convolution is filter-driven. For example, when "dropout" is used, there is usually no speedup. In contrast, the **data-driven** nature of OO visual processing makes it possible to achieve dramatic speedup on image locations where there is no object, since wherever there is no object, our system performs zero computation. We anticipate this feature to be very important to build the next generation computation efficient visual processing systems — there is no reason to perform more computation than needed.

8. Similar to the above points, convolution operates on a **dense** grid of features. It does not naturally supports **sparsity** — i.e., sparsity cannot be utilized to achieve speedup.

9. Finally, almost all hardware acceleration systems for vision rely on convolution. Object-oriented deep learning provide another a way to perform visual processing while capturing convolution as a special case. This computation paradigm may have some new properties that can be utilized to achieve more efficient hardware design. We have not explored this enough but believe this is a promising research direction in the future.

2 Disentanglement and Invariance Properties

With Object-Oriented deep learning, we can provide elegant solutions to a few difficult problems in vision. For example, the transformation disentanglement/invariance problem. Disentanglement is an appealing property of representations. Pooling or aggregation over the disentangled representations can give rise to various forms of invariances, which are quite useful in many tasks (such as invariant object recognition).

However, it is so far unclear how to learn such disentangled representations in deep networks. Object-oriented deep learning provides one solution: we can explicitly incorporate prior knowledge about some general transformations like rotation and scaling and make them as properties of the object.

3 Model

3.1 A General Object-oriented (OO) Layer

The input to a general OO layer is a (1-D) list of objects, their "signatures" and their properties. A "signature" is a (learned) vector that identifies the object and is invariant to the properties. The properties include features that are useful for performing intelligent tasks. They include but are not limited to positions (x,y), poses (scale, rotation, etc.), "objectness" (probability of being an object) pointers to its parts (for feature binding), and physical states (e.g., volume, mass, velocity, acceleration, etc.). So far we have experimented positions, poses and objectness (and part-whole relationships in some experiments). The output of the general OO layer is another 1-D list of objects and their properties.

As special cases of a general OO layer, we introduce the following two OO layers:

3.2 A Predicting/Voting OO Layer

In this layer, each input object will predict the existance of another set of objects and their properties. Each item of the 1-D list of objects will potentially generate more than one predicted objects. So the output list tends to be

longer than the input list.

It is a design choice how and how many output objects are predicted from each input object. In this work, each input object predicts A objects (where A = number of possible orientations) on 8 positions evenly distributed on a circle of radius R around the object plus 1 additional position at the center of the circle. For each predicted object, its signature is product of the signature of the predicting object and a weight matrix. Such weight matrix is different for each predicted position and orientation. Currently all objects share the same set of weight matrices for prediction. As for future research, we are exploring other predicting/voting approaches that will be more efficient and flexible.

3.3 A Binding OO Layer

This layer will try to solve a "binding problem" by merging relevant objects together. The output list tends to be shorter than the input list.

There could be many potential ways to perform binding, and it could be an interesting area of research. For computation efficiency, we adopt in this work a simple binding scheme: For each scale s, we choose N_s by M_s reference points evenly distributed on the image, each reference point has a receptive field of h (height) by w (width) by r (orientation space). We sum up the signature of all objects within the receptive field (RF) of each point, weighted by the distance of the object to the reference point divided by the radius of the RF (but unweighted version seem to work as well). The location (i.e., x,y, orientation) of the output objects are the centers of the reference points.

Optionally, we have some extra features that we have implemented and are currently experimenting: one can have a consistency-weighted binding scheme, where the "consistency" is computed by a normalized dot product (i.e., cosine distance, as commonly used in I-Theory [4, 5] and HMAX[6]) between the signature of the sum and that of each constituent object. This defines the object's contribution (or level of belonging/binding) to the aggregated object. The "objectness" of the aggregated object is defined to be the average of the absolute value of such normalized dot products — indicating how consistent the predictions are. We can sort the "objectness" of all reference points and keep the top p points as output objects. There could be other possible ways for predicting "objectness": for example, it can be predicted directly from each object's signature.

As another extension, we also have implemented an iterative binding scheme that refines the above "consistency weight" of each vote to each reference point by repeatly applying the above procedure: everytime the the output changes, the consistency-weights can be recomputed and vice versa. This can be done several iterations. Eventually, this clustering-like approach will determine which votes are consistent with each other, achieving a "feature binding" purpose. We expect this scheme to benefit visual recognition with clutterred/overlapping objects.

3.4 Biological Counterparts

When modeling the brain, a predicting/voting layer is similar to a process where a cell sends out predictions/signals to multiple spatial locations. A binding layer is similar to a process where a cell (actively) gathers nearby predictions/signals generated by other cells. In comparison, convolution is a cell-wise atomic operation, which glues these two processes together in a particular way that is very constrained.

Note that there is an important difference between our framework and biological neural system: the locations and receptive fields of cells in the brain are likely to be fixed. Our framework does not exclude the possibility that the locations of objects change from sample to sample, depending on the exact voting and binding algorithms one use. So our framework is more general than what the brain can do, but it of course remains a question whether this generality can achieve any advantage over the brain.

3.5 Nonlinearity, Batch Normalization

One can apply nonlinearity and batch normalization to the signature of objects at any stage. But usually we do it after the binding layer.

3.6 How to get first layer objects?

Now that we have described all the computations between lists of objects. One question remains: how do we get the lowest level objects in the first place.

Basically, each pixel location can be treated as containing an "pixel object" at any orientation — since its signature (r,g,b) is the same when viewed at any orientation. One just need to decide a grid of points to sample from and then sample arbitrary number of objects at arbitrary orientations from each pixel. The choice of grid may affect the performance: for rotation invariance experiment, we find that when sampling objects at orientation θ, a rotated grid at corresponding orientation θ gives better performance, perhaps giving less bias to a particular orientation.

When sampling objects at different scales, one resize the image to corresponding scales and then repeat the above processes. One should make sure the sampled positions correspond to the centers of the objects in the non-scaled coordinate system.

4 Experiments

Given the "exotic" nature of our framework, we cannot use any deep learning primitive like convolution. Instead, we implemented our framework from scratch using CUDA without using any existing deep learning framework or CuDNN (except batch normalization and some helper functions from MatConvNet [7]). Currently it is roughly 30% (very crude estimation) as fast as MatConvNet's CuDNN-enabled ConvNet (one scale one orientation, roughly same depth and number of features/parameters). It still has great potential to be speeded up. We plan to release the code soon with (yet another) deep learning framework, which has some mildly interesting features.

4.1 Standard Experiment on CIFAR-10

4.2 Settings

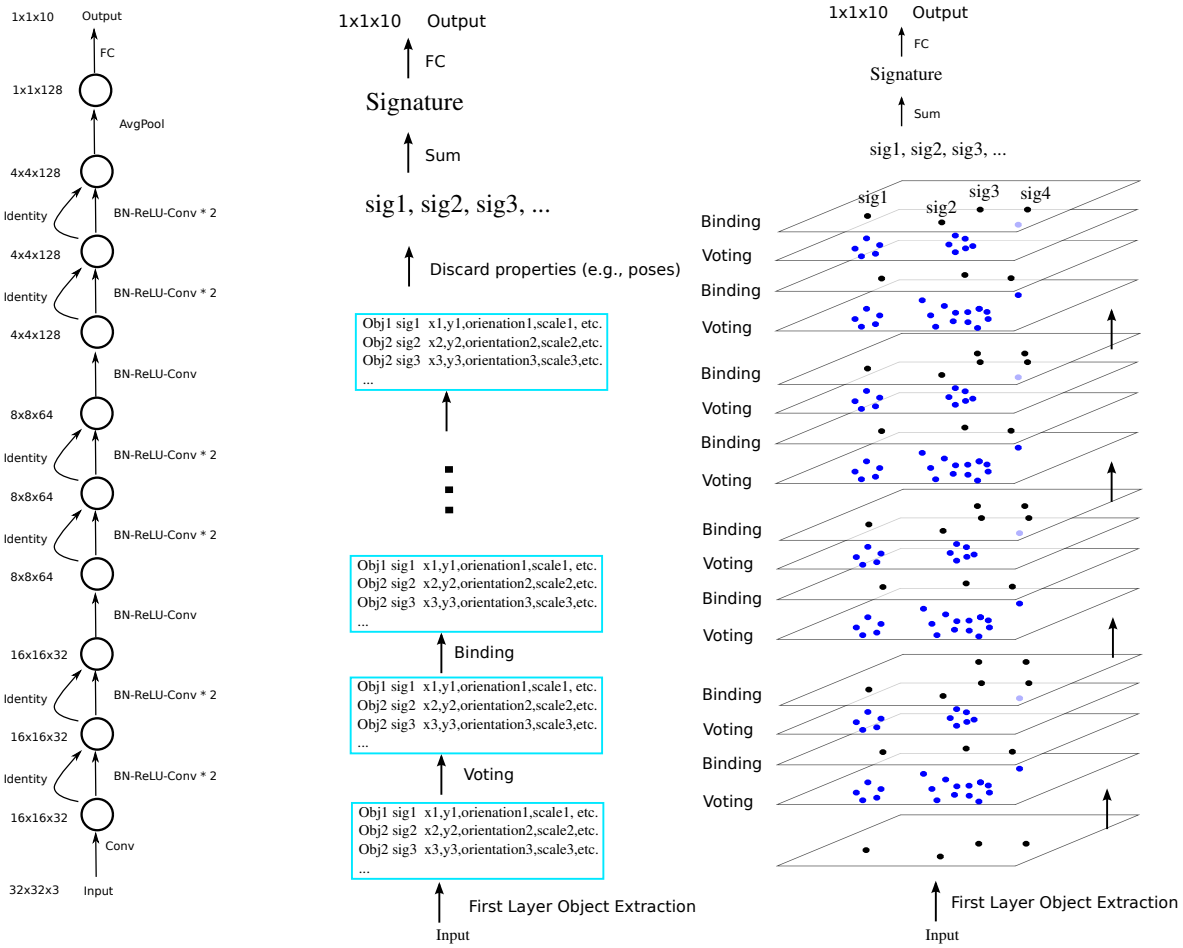
We tried our model on the standard CIFAR-10 dataset. A standard $\pm$ 2 pixel shift data augmentation is performed. The learning rate is 0.1 from epoch 1 to 40, and 0.01 from epoch 41 to 60. The batch size is 32. Following models are tried:

Simple ResNet As a baseline model, we tried a simple ResNet [8] shown in Figure 3 (A) in exactly the same training and testing settings.

Object-oriented Network (ONet) We used an object-oriented networks shown in Figure 3 (B) and (C). For this standard CIFAR-10 experiment, only one scale and orientation is used.

4.3 Results

As shown in Figure 4 and 5, we observe roughly same level of performance of ResNet and ONet on the CIFAR-10 dataset.



(A) ResNet (B) Object-oriented Network (C) Object-oriented Network (Visualization)

Figure 3: Visualizations of the architecture of ResNet and Object-Oriented Network that we use in our experiments. Note that we use RELU and batch normalization on the signatures of the objects after every **binding** layer (except the last one). "sig": signature, a vector that characterizes/identifies the object and is invariant to the properties of the object.

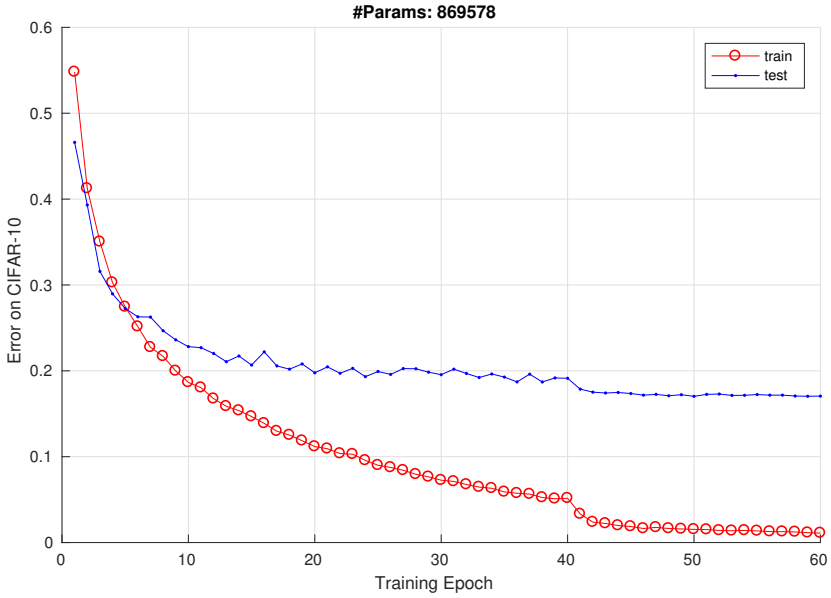


Figure 4: Train and test baseline ResNet on standard CIFAR-10. Note that this ResNet is not the most standard ResNet. Here the absolute performance is not very important — it serves mainly as a sanity check of our deep learning framework (implemented from scratch).

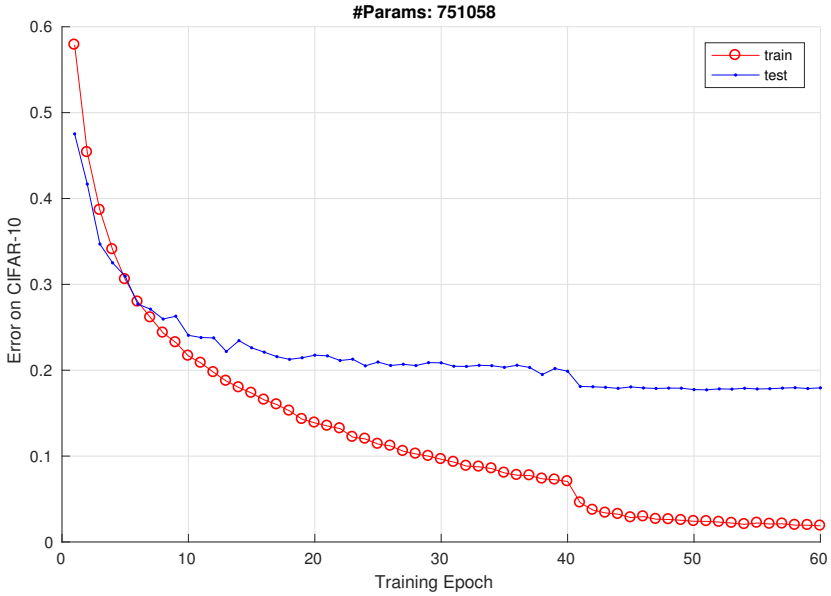


Figure 5: Train and test Object-Oriented Network on standard CIFAR-10. The performance ended up similar to that of ResNet (our baby version of ResNet).

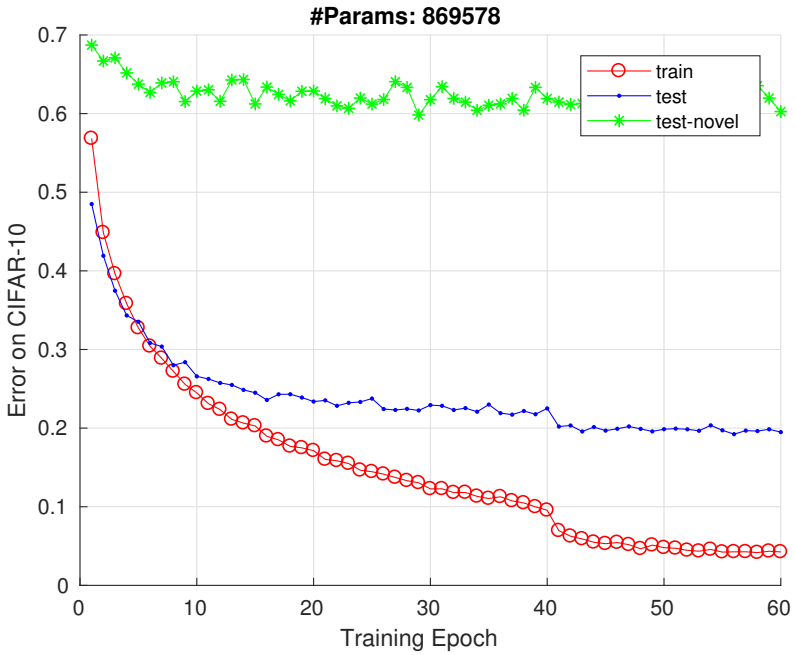


Figure 6: CIFAR-10: Train and test on rotation 0°. Test-novel: Test on novel rotations (all rotations from 0° to 360°). ResNet cannot generalize. Furthermore, we trained smaller models and confirmed that the poor generalization is not related to the number of parameters.

4.4 Generalize to Novel Rotations on CIFAR-10

4.5 Settings

Next, we perform a more challenging task: we train the network on one rotation and test it on 360 degrees of rotations on CIFAR-10. A standard $\pm$ 2 pixel shift data augmentation is performed. The learning rate is 0.1 from epoch 1 to 40, and 0.01 from epoch 41 to 60. The batch size is 32. Following models are tried:

Simple ResNet As a baseline model, we tried again the simple ResNet in exactly the same training and testing settings.

Object-oriented Network (ONet) We tried the same Object-oriented Networks on this task. Here we have 8 orientation bins in the binding layer so that every object can have one of the eight possible orientations. Objects of different orientations can also predict each other — this achieves a compact representation of orientations of objects and parts throughout the network.

4.6 Results

ResNet cannot generalize to novel rotations as shown in Figure 6. In contrast, Object-oriented network generalizes to novel rotations reasonably well (Figure 7).

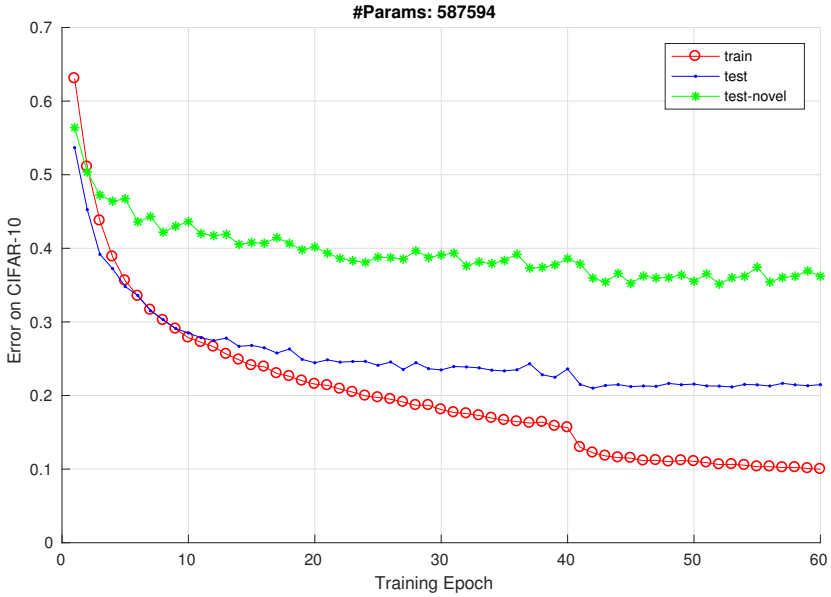


Figure 7: CIFAR-10: Train and test on rotation 0°. Test-novel: Test on novel rotations (all rotations from 0° to 360°). Object-oriented network generalizes much better than ResNet.

5 Related Work

This work was directly motivated by our CBMM's new research thrust of "Towards Symbols" and our previous work on computational modeling of ventral stream and invariant object recognition [4, 5, 9, 10]. We would like to see if one can build a symbolic computation framework that can support invariant object recognition, account for some observations of human vision (e.g., pose awareness, feature binding, etc.), and at the same time interpretable.

A further survey of the literature reveals several lines of exciting work that share some of our motivations and ideas from different perspectives.

Hinton has been working on a class of models called "capsules" [11]. Based on our limited understanding of "capsules", a "capsule" is a group of neurons at a fixed location of the network (thus having a fixed receptive field), dedicated to detecting one object at a time. A "capsule" layer would have a dense grid of capsules detecting objects at a grid of locations. In this sense, it is still very similar to traditional neural networks — each traditional neuron is just replaced by a small sub-network called "capsule". On the other hand, the object-oriented deep learning we propose is quite different in the sense that there is no fixed grid of anything[2]. In addition, the "generalized pose" in "capsules" is not the ordinary type of pose (like what we use), but rather a combination of pose and appearance (deformation, color, lighting, etc.).

Another line of work is a class of "physics engine" type of networks including Interaction Networks [12, 13], compositional Neural Physics Engine (NPE) [14], Relational Networks [15], etc., where some form of object representation (either hand-coded or extracted from pixels using ConvNet) is used as inputs to their networks. The main differences from our work are: 1. Their object representations are only for high-level processings (e.g., physics simulation, reinforcement learning, etc.) and are not intended as a replacement of ConvNet. 2. They hold a more conventional view of building high-level symbols from neural infrastructure, instead of symbolizing most basic neural computations.

Spatial Transformer Networks [16] try to achieve invariance to affine transformations by dynamically performing a warping on the image/feature space.

[2]The OO binding layer in this paper may have a grid but it is just a design choice for efficiency. We also are currently experimenting irregular and adaptive binding.

Regarding more flexible visual processing, we find there is an interesting work called "Deformable Convolutional Networks" [17], which offers more flexible convolutional filters. In comparison, we believe our framework is more expressive. Also our framework has additional implications on interpretability, sparsity, new hardware acceleration, invariance, disentangled representations, symbolism/conectionism, etc.

References

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[2] G. Neubig, C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Ballesteros, D. Chiang, D. Clothiaux, T. Cohn, *et al.*, "Dynet: The dynamic neural network toolkit," *arXiv preprint arXiv:1701.03980*, 2017.

[3] M. Looks, M. Herreshoff, D. Hutchins, and P. Norvig, "Deep learning with dynamic computation graphs," *arXiv preprint arXiv:1702.02181*, 2017.

[4] F. Anselmi, J. Z. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, and T. Poggio, "Unsupervised learning of invariant representations in hierarchical architectures," *arXiv preprint arXiv:1311.4158*, 2013.

[5] Q. Liao, J. Z. Leibo, and T. Poggio, "Learning invariant representations and applications to face verification," in *Advances in Neural Information Processing Systems*, pp. 3057–3065, 2013.

[6] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nat. Neurosci.*, vol. 2, no. 11, pp. 1019–1025, 1999.

[7] A. Vedaldi and K. Lenc, "Matconvnet: Convolutional neural networks for matlab," in *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pp. 689–692, ACM, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385v1 [cs.CV] 10 Dec 2015*, 2015.

[9] Q. Liao, J. Z. Leibo, Y. Mroueh, and T. Poggio, "Can a biologically-plausible hierarchy effectively replace face detection, alignment, and recognition pipelines?," *arXiv preprint arXiv:1311.4082*, 2013.

[10] Q. Liao, J. Z. Leibo, and T. Poggio, "Unsupervised learning of clutter-resistant visual representations from natural videos," *arXiv preprint arxiv:1409.3879, CBMM Memo n 23*, 2014.

[11] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*, pp. 44–51, Springer, 2011.

[12] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems*, pp. 4502–4510, 2016.

[13] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, "Visual interaction networks," *arXiv preprint arXiv:1706.01433*, 2017.

[14] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," *arXiv preprint arXiv:1612.00341*, 2016.

[15] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," *arXiv preprint arXiv:1706.01427*, 2017.

[16] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *Advances in Neural Information Processing Systems*, pp. 2017–2025, 2015.

[17] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," *arXiv preprint arXiv:1703.06211*, 2017.