arXiv:2212.11696v3 [cs.CV] 1 Feb 2023

REVERSIBLE COLUMN NETWORKS

Yuxuan Cai[1] **Yizhuang Zhou**[1] **Qi Han**[1] **Jianjian Sun**[1] **Xiangwen Kong**[1] **Jun Li**[1]
Xiangyu Zhang[12] *

MEGVII Technology[1]
Beijing Academy of Artificial Intelligence[2]
{caiyuxuan, zhouyizhuang, hanqi, zhangxiangyu}@megvii.com

ABSTRACT

We propose a new neural network design paradigm *Reversible Column Network (RevCol)*. The main body of RevCol is composed of multiple copies of subnetworks, named *columns* respectively, between which multi-level reversible connections are employed. Such architectural scheme attributes RevCol very different behavior from conventional networks: during forward propagation, features in RevCol are learned to be gradually *disentangled* when passing through each column, whose total information is maintained rather than *compressed* or discarded as other network does. Our experiments suggest that CNN-style RevCol models can achieve very competitive performances on multiple computer vision tasks such as image classification, object detection and semantic segmentation, especially with large parameter budget and large dataset. For example, after ImageNet-22K pre-training, RevCol-XL obtains 88.2% ImageNet-1K accuracy. Given more pre-training data, our largest model RevCol-H reaches **90.0%** on ImageNet-1K, **63.8%** AP_{box} on COCO detection minival set, **61.0%** mIoU on ADE20k segmentation. To our knowledge, it is the best COCO detection and ADE20k segmentation result among *pure* (static) CNN models. Moreover, as a general macro architecture fashion, RevCol can also be introduced into transformers or other neural networks, which is demonstrated to improve the performances in both computer vision and NLP tasks. We release code and models at https://github.com/megvii-research/RevCol

1 INTRODUCTION

Information Bottleneck principle (IB) (Tishby et al., 2000; Tishby & Zaslavsky, 2015) rules the deep learning world. Consider a typical supervised learning network as in Fig. 1 (a): layers close to the input contain more low-level information, while features close to the output are rich in semantic meanings. In other words, information unrelated to the target is gradually *compressed* during the layer-by-layer propagation. Although such learning paradigm achieves great success in many practical applications, it might not be the optimal choice in the view of *feature learning* – down-stream tasks may suffer from inferior performances if the learned features are *over* compressed, or the learned semantic information is irrelevant to the target tasks, especially if a significant domain gap exists between the source and the target tasks (Zamir et al., 2018). Researchers have devoted great efforts to make the learned features to be more universally applicable, *e.g.* via self-supervised pre-training (Oord et al., 2018; Devlin et al., 2018; He et al., 2022; Xie et al., 2022) or multi-task learning (Ruder, 2017; Caruana, 1997; Sener & Koltun, 2018).

In this paper, we mainly focus on an alternative approach: building a network to learn *disentangled representations*. Unlike *IB* learning, disentangled feature learning (Desjardins et al., 2012; Bengio et al., 2013; Hinton, 2021) does not intend to extract the most related information while discard the less related; instead, it aims to embed the task-relevant concepts or semantic words into a few decoupled dimensions respectively. Meanwhile the whole feature vector roughly maintains as much information as the input. It is quite analogous to the mechanism in biological cells (Hinton, 2021; Lillicrap et al., 2020) – each cell shares an identical copy of the whole genome but has different expression intensities. Accordingly in computer vision tasks, learning disentangled features is also

*Corresponding author. This work is supported by The National Key Research and Development Program of China (No. 2017YFA0700800) and Beijing Academy of Artificial Intelligence (BAAI).

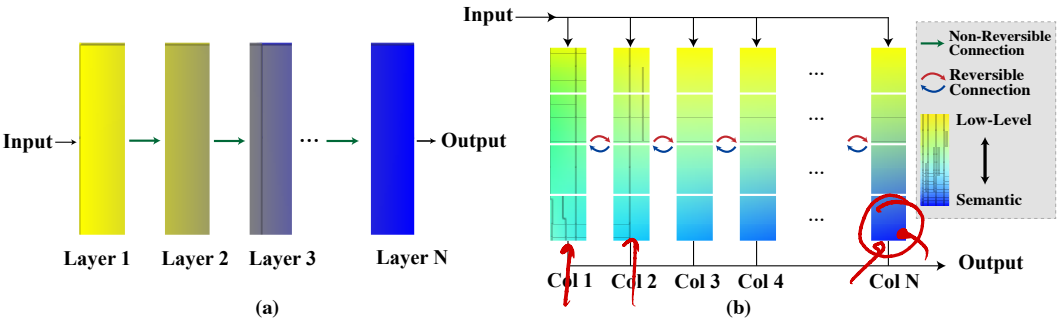


Figure 1: Sketch of the information propagation in: (a) Vanilla single-column network. (b) Our reversible column network. Yellow color denotes low-level information and blue color denotes semantic information.

reasonable: for instance, high-level semantic representations are tuned during *ImageNet* pre-training, meanwhile the low-level information (*e.g.* locations of the edges) should also be maintained in other feature dimensions in case of the demand of down-stream tasks like object detection.

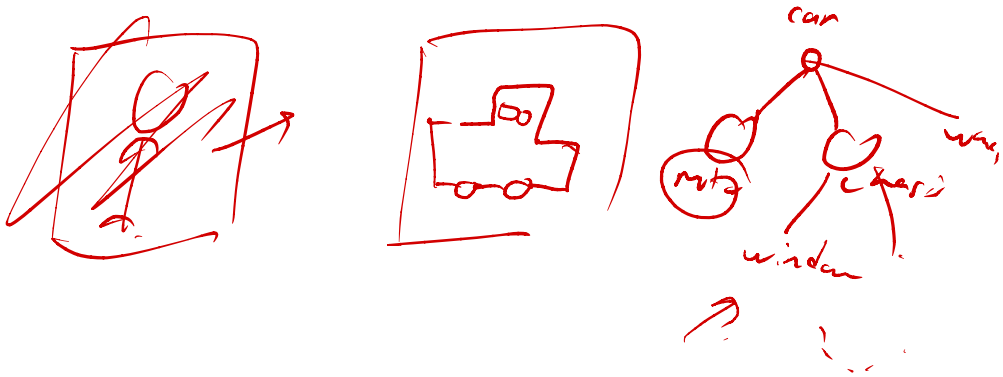
Fig. 1 (b) sketches our main idea: *Reversible Column Networks (RevCol)*, which is greatly inspired by the big picture of *GLOM* (Hinton, 2021). Our network is composed of N subnetworks (named *columns*) of identical structure (however whose weights are not necessarily the same), each of which receives a copy of the input and generates a prediction. Hence multi-level embeddings, *i.e.* from low-level to highly semantic representations, are stored in each column. Moreover, *reversible transformations* are introduced to propagate the multi-level features from i-th column to $(i+1)$-th column without information loss. During the propagation, since the complexity and nonlinearity increases, the quality of all feature levels is expected to gradually improve. Hence the last column (Col N in Fig. 1 (b)) predicts the final disentangled representations of the input.

In RevCol, one of our key contributions is the design of the *reversible transformations* between adjacent columns. The concept is borrowed from the family of *Reversible Networks* (Chang et al., 2018; Gomez et al., 2017; Jacobsen et al., 2018; Mangalam et al., 2022); however, conventional reversible structures such as *RevNets* (Gomez et al., 2017) (Fig. 2 (a)) usually have two drawbacks: first, feature maps within a reversible block are restricted to have the same shape[*]; second, the last two feature maps in RevNets have to contain both low-level and high-level information due to the reversible nature, which may be difficult to optimize as in conflict with *IB* principle. In this paper, we overcome the drawbacks by introducing a novel reversible multi-level fusion module. The details are discussed in Sec. 2.

We build a series of CNN-based *RevCol* models under different complexity budgets and evaluate them in mainstream computer vision tasks, such as *ImageNet* classification, *COCO* object detection and instance segmentation, as well as *ADE20K* semantic segmentation. Our models achieve comparable or better results than sophisticated CNNs or *vision transformers* like ConvNeXt (Liu et al., 2022b) and Swin (Liu et al., 2021). For example, after ImageNet-22K pre-training, our RevCol-XL model obtains **88.2%** accuracy on ImageNet-1K without using transformers or large convolutional kernels (Ding et al., 2022b; Liu et al., 2022b; Han et al., 2021). More importantly, we find RevCol can scale up well to large models and large datasets. Given a larger private pre-training dataset, our biggest model RevCol-H obtains **90.0%** accuracy on ImageNet-1K classification, **63.8%** AP_{box} on COCO detection minival set, and **61.0%** mIoU on ADE20K segmentation, respectively. To our knowledge, it is the best reversible model on those tasks, as well as the best *pure* CNN model on COCO and ADE20K which only involves static kernels without dynamic convolutions (Dai et al., 2017; Ma et al., 2020). In the appendix, we further demonstrate RevCol can work with transformers (Dosovitskiy et al., 2020; Devlin et al., 2018) and get improved results on both computer vision and NLP tasks. Finally, similar to RevNets (Gomez et al., 2017), RevCol also shares the bonus of memory saving from the reversible nature, which is particularly important for large model training.

Relation to previous works. Although our initial idea on feature disentangling is derived from *GLOM* (Hinton, 2021), in *RevCol* there are a lot of simplifications and modifications. For example,

[*]In precise, feature maps of odd and even indexes should be equal sized respectively.



arXiv:2102.12627v1 [cs.CV] 25 Feb 2021

How to represent part-whole hierarchies in a neural network

Geoffrey Hinton
Google Research
&
The Vector Institute
&
Department of Computer Science
University of Toronto

February 22, 2021

Abstract

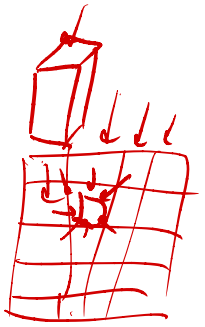
This paper does not describe a working system. Instead, it presents a single idea about representation which allows advances made by several different groups to be combined into an imaginary system called GLOM[1]. The advances include transformers, neural fields, contrastive representation learning, distillation and capsules. GLOM answers the question: How can a neural network with a fixed architecture parse an image into a part-whole hierarchy which has a different structure for each image? The idea is simply to use islands of identical vectors to represent the nodes in the parse tree. If GLOM can be made to work, it should significantly improve the interpretability of the representations produced by transformer-like systems when applied to vision or language.

1 Overview of the idea

There is strong psychological evidence that people parse visual scenes into part-whole hierarchies and model the viewpoint-invariant spatial relationship between a part and a whole as the coordinate transformation between intrinsic coordinate frames that they assign to the part and the whole [Hinton, 1979]. If we want to make neural networks that understand images in the same way as people do, we need to figure out how neural networks can represent part-whole

[1]GLOM is derived from the slang "glom together" which may derive from the word "agglomerate".

hierarchies. This is difficult because a real neural network cannot dynamically allocate a group of neurons to represent a node in a parse tree[2]. The inability of neural nets to dynamically allocate neurons was the motivation for a series of models that used "capsules" [Sabour et al., 2017, Hinton et al., 2018, Kosiorek et al., 2019]. These models made the assumption that a group of neurons called a capsule would be permanently dedicated to a part of a particular type occurring in a particular region of the image. A parse tree could then be created by activating a subset of these pre-existing, type-specific capsules and the appropriate connections between them. This paper describes a very different way of using capsules to represent the part-whole hierarchy in a neural net.



Even though this paper is primarily concerned with the perception of a single static image, GLOM is most easily understood as a pipeline for processing a sequence of frames, so a static image will be treated as a sequence of identical frames.

The GLOM architecture[3] is composed of a large number of columns[4] which all use exactly the same weights. Each column is a stack of spatially local autoencoders that learn multiple levels of representation for what is happening in a small image patch. Each autoencoder transforms the embedding at one level into the embedding at an adjacent level using a multilayer bottom-up encoder and a multilayer top-down decoder. These levels correspond to the levels in a part-whole hierarchy. When shown an image of a face, for example, a single column might converge on embedding vectors[5] representing a nostril, a nose, a face, and a person. Figure 1 shows how the embeddings at different levels interact in a single column.

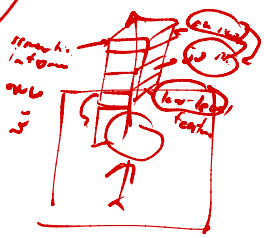


Figure 1 does not show the interactions between embeddings at the same level in different columns. These are much simpler than the interactions within a column because they do not need to implement part-whole coordinate transforms. They are like the attention-weighted interactions between columns representing different word fragments in a multi-headed transformer [Devlin et al., 2018], but they are simpler because the query, key and value vectors are all identical to the embedding vector. The role of the inter-column interactions is to produce islands of identical embeddings at a level by making each embedding vector at that level regress towards other similar vectors at nearby locations. This creates multiple local "echo chambers" in which embeddings at a level attend mainly to other like-minded embeddings.

[2]What neurons do is determined by their incoming and outgoing weights and real neurons cannot completely change these weights rapidly.

[3]The GLOM architecture has some similarity to models that use the errors in top-down predictions as their bottom-up signals [Rao and Ballard, 1999], but in a nonlinear system the bottom-up signals cannot just carry the prediction error because the full activity vector is required to select the right operating regime for the non-linear units.

[4]Each level in a column bears some resemblance to a hypercolumn as described by neuroscientists.

[5]An embedding vector is the activity vector of a capsule.

At each discrete time and in each column separately, the embedding at a level is updated to be the weighted average of four contributions:

1. The prediction produced by the bottom-up neural net acting on the embedding at the level below at the previous time.
2. The prediction produced by the top-down neural net acting on the embedding at the level above at the previous time.
3. The embedding vector at the previous time step.
4. The attention-weighted average of the embeddings at the same level in nearby columns at the previous time.

For a static image, the embeddings at a level should settle down over time to produce distinct islands of nearly identical vectors. These islands should be larger at higher levels as shown in figure 2. Using the islands of similarity to represent the parse of an image avoids the need to allocate groups of neurons to represent nodes of the parse tree on the fly, or to set aside groups of neurons for all possible nodes in advance. Instead of allocating neural hardware to represent a node in a parse tree and giving the node pointers to its ancestor and descendants, GLOM allocates an appropriate activity vector to represent the node and uses the same activity vector for all the locations belonging to the node[6]. The ability to access the ancestor and descendants of the node is implemented by the bottom-up and top down neural nets rather than by using RAM to do table look-up.

Like BERT [Devlin et al., 2018], the whole system can be trained end-to-end to reconstruct images at the final time-step from input images which have missing regions, but the objective function also includes two regularizers that encourage islands of near identical vectors at each level. The regularizers are simply the agreement between the new embedding at a level and the bottom-up and top-down predictions. Increasing this agreement facilitates the formation of local islands.

[6]The idea of using similarity of vectors to do segmentation has been used in earlier work on directional unit Boltzmann machines [Zemel et al., 1995]

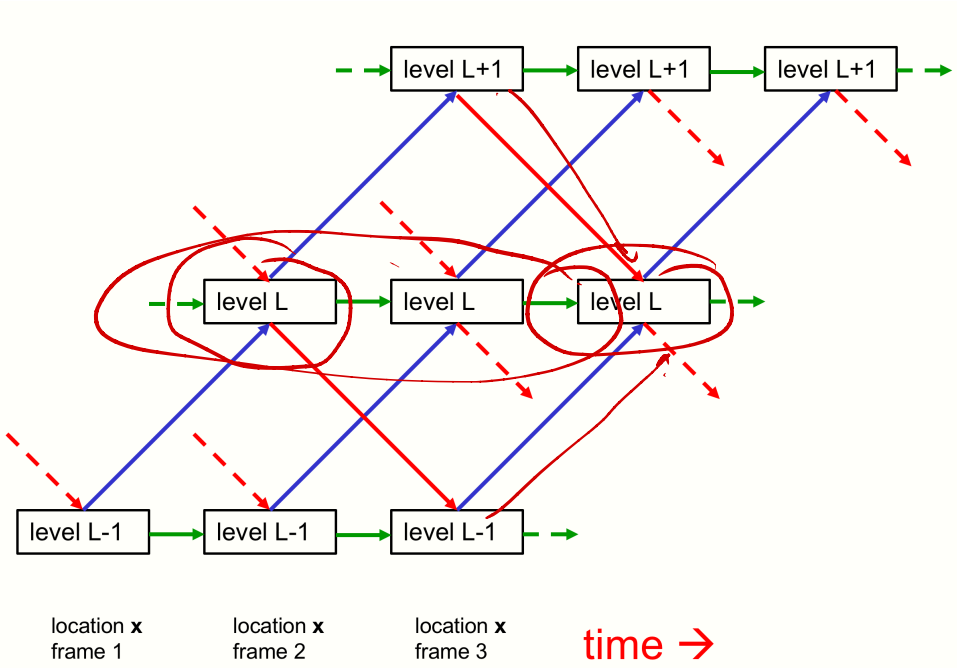


Figure 1: Showing the bottom-up, top-down, and same level interactions among three adjacent levels of the proposed GLOM architecture *for a single column*. The blue and red arrows representing bottom-up and top-down interactions are implemented by two different neural networks that have several hidden layers. These networks can differ between pairs of levels but they are shared across columns and across time-steps. The top-down net should probably use sinusoidal units[Sitzmann et al., 2020]. For a static image, the green arrows could simply be scaled residual connections that implement temporal smoothing of the embedding at each level. For video, the green connections could be neural networks that learn temporal dynamics based on several previous states of the capsule. Interactions between the embedding vectors at the same level in different columns are implemented by a non-adaptive, attention-weighted, local smoother which is not shown.

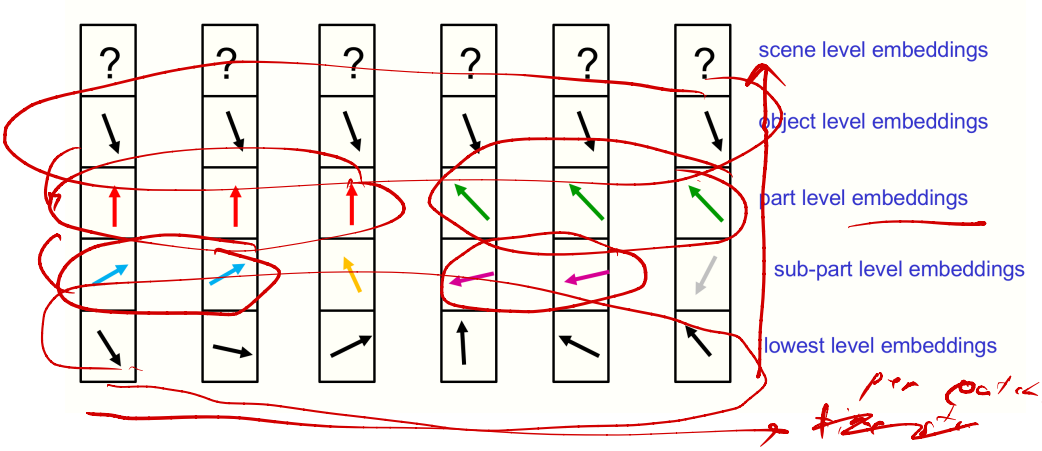


Figure 2: A picture of the embeddings *at a particular time* in six nearby columns. All of the locations shown belong to the same object and the scene level has not yet settled on a shared vector. The complete embedding vector for each location is shown by dividing the vector into a separate section for each level in the part-whole hierarchy and then showing the high-dimensional embedding vector for a level as a 2-D vector. This makes it easy to illustrate alignment of the embedding vectors of different locations. The islands of identical vectors at the various levels shown in the figure represent a parse tree. But islands of identity are considerably more powerful than phrase structure grammars. They have no difficulty representing disconnected objects as in "Will this slow phrase structure grammarians down?"

2 Introduction

This paper proposes the idea of using islands of similar vectors to represent the parse tree of an image and then explores some of the many ramifications of this idea by describing an imaginary system called GLOM that implements it. It concludes with some speculations about how the brain might implement some aspects of GLOM. But first some disclaimers:

Disclaimer1: Human vision is a sampling process in which intelligently chosen fixation points are used to acquire the information required to perform a task using retinas that have much higher resolution around the fixation point. The same neural circuitry is reused for each new fixation. For the purposes of this paper, I assume a single retina or camera with uniform resolution and only consider what happens on the first fixation.

Disclaimer 2: To avoid cumbersome terms like "sub-sub-parts", I will often talk about parts and wholes as if there were only two levels in the part-whole hierarchy. But a section of the complete embedding vector that is called a whole when considering levels L-1 and L is also called a part when considering levels L and L+1.

In a computer that has general purpose, random access memory, the obvious way to represent the part-whole hierarchy for a specific image is to create a graph structure for that particular image by dynamically allocating pieces of the memory to the nodes in the graph and giving each node pointers to the nodes it is connected to. Combining this type of dynamically created graph with neural network learning techniques has recently shown great promise [Bear et al., 2020], but if the whole computer is a neural network, it is far less obvious how to represent part-whole hierarchies that are different for every image if we want the structure of the neural net to be identical for all images. If we allow three-way interactions in which the activity of one neuron gates the connection between two other neurons [Hinton, 1981c], it is easy to make the connections dynamic, but it is still unclear how to dynamically create a graph structure without the ability to allocate neurons on the fly. It is especially difficult in a real neural net where the knowledge is in the connection weights, which cannot easily be copied.

One rather cumbersome solution to this problem is to set aside a group of neurons, called a capsule, for each *possible* type of object or part in each region of the image[7] and to use a routing algorithm to dynamically connect a small subset of active capsules into a graph that represents the parse of the image at hand. The activities of neurons within a capsule can then represent properties of a part such as the pose or deformation of a particular mouth or face.

With considerable effort, models that use capsules have achieved some suc-

[7] These regions can be larger for higher level parts which are more diverse but occur more sparsely in any one image.

cesses in supervised and unsupervised learning on small datasets [Sabour et al., 2017, Hinton et al., 2018, Kosiorek et al., 2019], but they have not scaled well to larger datasets [Barham and Isard, 2019]. Capsules do not have the signature of really practical ideas like stochastic gradient descent or transformers which just *want* to work. The fundamental weakness of capsules is that they use a mixture to model the set of possible parts. This forces a hard decision about whether a car headlight and an eye are really different parts. If they are modeled by the same capsule, the capsule cannot predict the identity of the whole. If they are modeled by different capsules the similarity in their relationship to their whole cannot be captured.

One way to avoid using a mixture for modeling the different types of part is to have a set of identical, "universal" capsules, each of which contains enough knowledge to model any type of part [Locatello et al., 2020, Srivastava et al., 2019, Sun et al., 2020b]. This allows part identities to have distributed representations, which allows better sharing of knowledge between similar parts. In neuroscience terminology, identities are value-coded rather than place-coded. However, it creates a symmetry breaking problem in deciding which universal object-level capsule each part should be routed to[8].

A more radical version of universal capsules, which avoids both symmetry breaking and routing, is to pre-assign a universal capsule to every location in the image. These ubiquitous universal capsules can be used to represent whatever happens to be at that location. An even more profligate version is to dedicate several different levels of ubiquitous universal capsule to each location so that a location can belong to a scene, an object, a part and a sub-part simultaneously. This paper explores this profligate way of representing the part-whole hierarchy. It was inspired by a biological analogy, a mathematical analogy, and recent work on neural scene representations [Ha, 2016, Sitzmann et al., 2019].

2.1 The biological analogy

All the cells in the body have a copy of the whole genome. It seems wasteful for brain cells to contain the instructions for behaving like liver cells but it is convenient because it gives every cell its own private access to whatever DNA it might choose to express. Each cell has an expression intensity[9] for each gene and the vector of expression intensities is similar for cells that form part of the same organ.

The analogy with neural nets goes like this: Each location in the image corresponds to a biological cell. The complete embedding vector for a location is like the vector of gene expression intensities for a cell. The forward pass is like

[8] Adam Kosoriek suggested using universal capsules in 2019, but I was put off by the symmetry breaking issue and failed to realise the importance of this approach.

[9] I use the word "intensity" rather than the word "level" so as not to confuse scalar intensities with discrete levels in a part-whole hierarchy.

the developmental process that allows a new vector of gene expression intensities to be determined by the previous vectors of expression intensities. Objects are like organs: They are a collection of locations whose embedding vectors are all very similar at a high level. Within an object, the embedding vectors may differ at lower levels that correspond to the parts of the object (see figure 2).

2.2 The mathematical analogy

The Kolmogorov-Arnold superposition theorem states that every multivariate continuous function can be represented as a superposition of continuous functions of one variable[10]. For example, multiplication can be represented as the sum of the logs of the individual arguments followed by exponentiation. In machine learning terminology, when it comes to multi-argument functions, addition is all you need. This assumes, of course, that you can find the right single-argument functions to encode the arguments of the multivariate function you want to represent and then find the right function to decode the sum. Kolmogorov proved this can always be done but the encoder functions used for the proof are so bizarre that they are of no practical relevance to neural networks.

The theorem does, however, suggest an interesting approach to combining information coming from many different locations. Perhaps we can learn how to encode the information at each location in such a way that simply averaging the encodings from different locations is the only form of interaction we need[11]. This idea is already used in set transformers [Lee et al., 2019] for combining information from different members of a set. If we modify this suggestion slightly to use an attention-weighted local average, we get a particularly simple form of transformer in which the key, the query and the value are all the same as the embedding itself and the only interaction between locations is attention-weighted smoothing at each level. All of the adaptation occurs in the bottom-up and top-down neural networks at each location, which are depicted by the blue and red arrows in figure 1. These networks are shared across all locations and all time-steps, but possibly not across all levels of the part-whole hierarchy.

2.3 Neural fields

Suppose we want to represent the value of a scalar variable, such as the depth or intensity, at every point in an image. A simple way to do this is to quantize the image locations into pixels and use an array that specifies the scalar variable at each pixel. If the values of different pixels are related, it may be more efficient

[10] This solves a version of Hilbert's 13th problem.

[11] This has a resemblance to variational learning [Neal and Hinton, 1997], where we start by assuming that the log posterior distribution over the whole set of latent variables is determined by the sum of their individual log posterior distributions and then we try to learn a model for which this additive approximation works well.

to use a neural network that takes as input a code vector representing the image and outputs an array of pixel values. This is what the decoder of an autoencoder does. Alternatively we could use a neural network that takes as input a code vector representing the image plus an additional input representing an image location and outputs the predicted value at that location. This is called a neural field [12] and this way of using neural networks has recently become very popular [Ha, 2016, Sitzmann et al., 2020, Mildenhall et al., 2020]. Figure 3 shows a very simple example in which the intensities at a set of locations can all be reconstructed from the same code, even though the intensities vary.

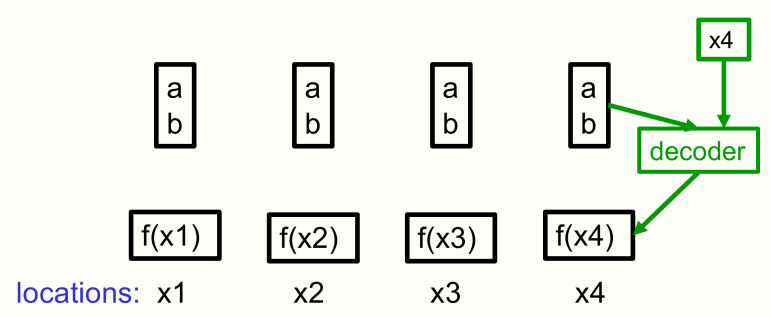


Figure 3: A very simple example of a neural field using individual pixels as the locations. The intensities of four pixels can all be represented by the same code (a, b) even though their intensities vary according to the function $f(x) = ax + b$. The decoder has an extra input which specifies the location.

For a complicated image, the neural net could transform a code vector representing the whole image plus a vector representing an image location into the value at that location. But if images are composed of familiar objects and objects are composed of familiar parts it is much more efficient to use a hierarchy of neural fields[13]. In GLOM, the scene-level top-down neural network converts the scene vector plus an image location into the appropriate object vector for that location. This vector includes information about the 3-D pose of the object relative to the camera. All of the locations that belong to the same object are given exactly the same object-level vector. The object-level top-down neural network then converts an object vector plus a location into the part vector that is appropriate for that location and so on. For example, exactly the same top-

[12] An early example of using neural fields is described in [Oore et al., 1997]. The "image" is always the same, so only the location input is needed. The 12 channels at each "image" location are the depths returned by 12 sonar detectors pointing in different directions. The match between the neural net's prediction for each location and the actual data at the robot's current location is used to perform a Bayesian update of the mobile robot's probability distribution over locations.

[13] A small step in this direction is simply to have a separate neural field for each type of object. This makes it easy to represent scenes composed of familiar objects in novel arrangements [Niemeyer and Geiger, 2020].

down network acting on exactly the same face vector is able to predict the nose vector in some locations and the mouth vector in other locations.

2.4 Explicit versus emergent representations of the part-whole hierarchy

In the symbolic AI representation of the part-whole hierarchy, each node has a unique symbol or memory address, and this symbol or address has an arbitrary relationship to the content. In GLOM, the embedding vector at a particular level that is shared by all the locations in an island at that level plays the same role as the address of a node in a graph, but this vector is not arbitrary. The top-down neural network must predict the embedding vector of a part at level L from the embedding vector of an object at level L+1. This involves using the pose relative to the camera encoded at level L+1 and a representation of the image location to compute where the location is within the intrinsic coordinate frame of the object. This determines which level L part the location belongs to.

There is a very important difference between using an address bus to follow a pointer to the representation of a part of a known object and using a top-down neural network to compute the part vector. Unlike table-lookup, the top-down neural net finds it much easier to deal with shapes in which there are symmetries between the parts. Replicated parts, like the legs of a centipede for example, add very little complexity to the neural net and this remains true even if the legs change along the centipede, so long as they change in a predictable way. Bilateral symmetries that align with the intrinsic coordinate frame of an object reduce the required capacity of the top-down neural net by almost a factor of two.

It is much harder, however, for the neural net to make use of symmetries that do not align with the coordinate frame[14], and people are generally unaware of such symmetries. Most people, for example, are totally unaware of the threefold rotational symmetry of a cube, despite its name, until they are forced to use a body diagonal through the center of the cube as its intrinsic top-bottom axis [Hinton, 1979]. They then cease to be aware of any of the right angles in the cube because these angles no longer align with the new intrinsic coordinate frame[15].

[14] This is why Canonical Capsules [Sun et al., 2020b] discover the natural intrinsic coordinate frames.

[15] Most people have enormous difficulty imagining a cube when they are forced to use a body diagonal as the top-bottom axis. When asked to point out the corners that are not at the two ends of this axis, they typically point out four corners arranged in a square about halfway up the axis. This structure (two square-based pyramids stuck together base-to-base is actually an octahedron. An octahedron is the dual of a cube with vertices for faces and faces for vertices. So people preserve the fourfold rotational symmetry of a cube relative to its familiar coordinate system. This suggests that the symmetry structure is one of the most important properties encoded in the embedding vector of an object.

3 Some design decisions

This section discusses some decisions that need to be made when specifying the GLOM architecture.

3.1 How many levels are there?

GLOM assumes that the part-whole hierarchy has a fixed depth. People can deal with very deep hierarchies that have stars near the top and atomic nuclei near the bottom. The way to handle such an enormous range is to have a flexible mapping between entities in the world and the levels of GLOM [Hinton, 1990]. This allows the very same neurons to be used for stars at one time and for atomic nuclei at another, which has the added advantage of facilitating analogies between structures at very different scales like solar systems and atoms. The recursive re-use of the neural hardware raises many interesting issues about temporary storage and control flow [Ba et al., 2016] that will only be fleetingly addressed here.

A reasonable number of embedding levels would be about five. This allows for the pupil and the white of an eye to be the lowest-level sub-sub-parts in a scene composed of objects that are people with parts that are faces and sub-parts that are eyes. If finer details such as the dilation of the iris are required, people probably need to remap the world onto their hardware so that, for example, the face becomes the scene[16].

One interesting question is whether the bottom-up and top-down neural nets can be shared across levels as well as across locations. This would not work for the lower levels of language processing where entities at different levels, like phonemes or words, have very different properties, but vision is far more fractal. A big advantage of sharing across levels is that the vector representation used for a particular face when it was at the object level would then be forced to be consistent with its representation when it was at the part level. This would make it much easier to remap the visual world onto the hardware by simply copying all of the vectors up or down a few levels. After having used fine details of an eye to extract a highly informed vector representation of the eye when it was at the object level, this same vector could then be used to represent the eye when it was at the sub-part level[17].

[16] The levels in the part-whole hierarchy that are represented in the infero-temporal pathway are probably not the brain's only representation of space. The infero-temporal pathway is used for object recognition and there may well be other representations of the world that are used for other purposes such as detecting ego-motion or visually maintaining balance.

[17] This assumes that the vector can be transported to a different column if the fixation point changes when the face becomes the object of attention rather than the eye.

3.2 How fine-grained are the locations?

Locations could be as fine-grained as pixels, or they could correspond to larger image patches [Dosovitskiy et al., 2020]. To avoid additional complexity when explaining the basic idea of the paper, I will assume that the grid of locations remains the same at all levels, but this is probably not the best choice.

The granularity could change at different embedding levels. If higher levels in the part-whole hierarchy use a larger stride, the top-down neural net would need to output multiple different predictions for the multiple lower level locations that fall within one higher level location. Similarly, the bottom-up neural net would need to look at all the lower-level locations that get combined at the next level up.

One convenient way to be sensitive to a large spatial context whilst also being able to see fine detail is to have images at several different spatial resolutions all of which have the same number of pixels. The coarsest image conveys a large spatial context but lacks fine detail and the finest image conveys the fine details, but only for a small region. If the visual input is structured into multiple images in this way, it would make sense to make peripheral locations cover larger regions, but this paper will ignore that issue because it makes everything more complicated.

3.3 Does the bottom-up net look at nearby locations?

Even if the granularity at different levels remains unchanged, the bottom-up neural net could look at the embedding vectors at nearby locations. This is a less pure version of GLOM which allows the interactions between locations to be more complex than just averaging. The purely bottom-up pathway then resembles a convolutional neural net but with the predictions for the next level up being made by a multi-layer neural net that implements a far more complicated function than just a matrix multiply followed by a scalar non-linearity.

The disadvantage of allowing the bottom-up net to look at other locations is that two locations with identical representations at the part level may have different spatial contexts. We would then lose a very nice property of the pure version of GLOM: locations that have identical representations at the part level make exactly the same bottom-up predictions at the object level.

By looking at other locations, the bottom-up net can reduce the uncertainty before it predicts a distribution at the next level up and this seems like a good thing to do. But it should be possible to get a similar reduction in uncertainty *after* making the prediction when the attention-weighted smoothing combines an uncertain bottom-up prediction from one location with the uncertain bottom-up predictions from nearby locations. Of course, this assumes that the bottom-up net can represent the uncertainty in its predictions and that the uncertainties in

different locations can be combined correctly by the attention-weighted smoothing. This issue is addressed in section 9.

3.4 How does the attention work?

One of the contributors to the update of the embedding of level L at location x is the attention-weighted average of the embeddings of level L at nearby locations at the previous time step. GLOM assumes the simplest form of attention weighting in which the weight w_{xy} that location x gives to the embedding at location y is given by

$$w_{xy} = \frac{e^{\beta L_x . L_y}}{\sum_z e^{\beta L_x . L_z}} \tag{1}$$

where . is the scalar product of the two embedding vectors, z indexes all the locations that location x attends to at level L and β is an "inverse temperature" parameter that determines the sharpness of the attention. β could increase as GLOM settles to a firm interpretation of the image. The way attention is intended to work in GLOM has already been used successfully in a system called "ACNe" [Sun et al., 2020a].

Pioneering work on using Markov Random Fields for image segmentation [Geman and Geman, 1984] used the presence of a boundary between pixel x and pixel y to prevent x from attending to y. A boundary is more than just a big intensity difference between x and y because its existence depends on the intensities at other locations. Similarly, early work on learning spatially coherent properties of images used the presence of boundaries to select which expert interpolator to use [Becker and Hinton, 1993]. Like the seashore, boundaries have a rich life of their own and much more work needs to be done to integrate them into GLOM, especially into its attention mechanism.

3.5 The visual input

In most neural networks, the visual input arrives at the bottom layer. In GLOM, a patch of the raw visual input could define the bottom-level embedding at a location by vectorizing the intensities in the image patch, but it is probably more sensible to first apply a convolutional neural net that can see a larger region of the image. The output of this convolutional net would then be the primary, lowest level embedding at each location.

The convolutional net is an open loop way to solve the following inference problem: What lowest-level embedding for that location would be able to reconstruct the pixel intensities using the learned neural field shared by all locations. Once the lowest-level embedding has been initialized, it can be refined in a closed loop by backpropagating the reconstruction error through the neural field [Williams et al., 1995].

There is no need to confine the direct visual input to the primary embedding layer. A coarser scale convolutional net operating on a lower resolution image could provide useful hints about the higher-level embeddings. For example, a pale vertical oval with a darker horizontal band slightly more than halfway up suggests one kind of face [Viola and Jones, 2004] so a convolutional net operating on coarse pixels can provide useful information to directly initialize the higher-level embeddings[18].

4 Color and texture

Consider an object whose individual parts are either entirely pale green or entirely mauve. The color of a part is straightforward, but what color is the whole object? One of the motivations for GLOM was the idea that the whole object has a compound color which might be called "pale-green-or-mauve" and *at the object level* every location belonging to the object has exactly the same, compound color. The object is pale-green-and-mauve all over. When deciding which other locations at the object level to attend to, preference would be given to locations with a similar compound color.

A similar idea applies to textures. The individual texture elements have their own shapes and poses and spatial relationships, but an object with a textured surface has exactly the same texture everywhere *at the object level*. GLOM extends this idea to shapes. An object may have parts that are very different from one another, but at the object level it has exactly the same compound shape in all of the locations that it occupies.

5 Cluster discovery versus cluster formation

The EM capsule model [Hinton et al., 2018] attempts to activate capsules that represent wholes (e.g. a face) by looking for clusters of similar vector votes for the pose of the whole. These vector votes come from already identified parts (e.g. a nose or mouth) and although the weights on these votes can be modified by an iterative routing procedure the vector votes themselves remain fixed. This is quite problematic if one of the parts has an under-determined pose. For example, a circle representing an eye has no specific orientation and its position in a face depends on whether it is a left or right eye. It does, however, provide some information about the scale of the face and it makes a unimodal prediction for the location of the face in the direction orthogonal to the unknown line between the two eyes[19].

[18] The visual part of the thalamus has direct connections to multiple different levels in the hierarchy of visual areas.

[19] The Stacked Capsule Autoencoder model [Kosiorek et al., 2019] deals with this issue by using a set transformer [Lee et al., 2019] to allow the parts to interact. This should allow the

In GLOM, the embedding vector of a location at level L-1 does not cast an immutable vector vote for the embedding at level L. Instead, it provides a bottom-up vector contribution to this embedding that is combined with the vector contribution coming from level L+1 and the attention-weighted contributions coming from the level L embeddings of other locations to determine the updated level L embedding vector. The bottom-up contribution can start off being quite vague and it can become sharper from time-step to time-step as top-down and lateral contextual information progressively refines the level L-1 embedding of the location. The islands of similar embedding vectors at a level can be viewed as clusters, but these clusters are not *discovered* in immutable data. They are *formed* by the interaction between an intra-level process that favors islands of similarity and dynamically changing suggestions coming from the location's embeddings at adjacent levels.

6 Replicating embedding vectors over locations

At first sight, it seems very inefficient to give a copy of the object-level embedding vector to every location that belongs to an object. Compelling intuitions that stem from programming computers with random access memory suggest that it would be much better to have a single copy of a data-structure for the object. These intuitions are probably misleading for neural nets that do not have RAM, and even if RAM is available there are two good reasons for replicating the embedding vectors over an island.

The island growing process at each level may eventually settle down to several islands of near identical vectors, but the search for these islands needs to be able to consider alternative clusterings of locations into islands and it also needs to allow for negotiations between locations within an island about what identical vector to settle on at each level. These negotiations are non-trivial because each location is also trying to satisfy inter-level constraints that come from its own embedding vectors at the level above and the level below and these embeddings are also being refined at every time-step. During the search, it is very helpful for every location to have its own version of the embedding vector at each level. Uncertainty in the clustering can be represented by making the embedding vector at a location be a blend of the vectors for the different clusters that it might decide to join. This blend can be refined over time and the fact that it lives in a high-dimensional continuous space should make optimization easier.

Intuitively, a blend of two rather different embedding vectors is not similar to either vector. This is true in a low-dimensional vector space, but intuitions derived from low-dimensional spaces cannot be trusted when dealing with high-

poses and identities of the parts to be disambiguated before they attempt to activate capsules at the next level up.

dimensional spaces. The average of two high-dimensional vectors is much closer to each of those vectors than it is to a random vector. This can be understood by thinking about the correlation between the components of a vector and the components of its average with some other random vector. If the vectors are high-dimensional, this correlation will be very significant [20].

A further advantage of islands of near identity is that it allows long range interactions within a level to be sparse. If there is more sparsity at higher levels, the interactions can be longer range without increasing the amount of computation. For locations that belong to an island far away, all the object-level information about that island is contained in each of its locations, so it is only necessary to sample one of those locations for that distant island to compete with other closer islands for a location's attention. Of course, this means that distant islands contribute fewer logits to the attention softmax than closer islands, but the exponential used in the attentional softmax means that one logit from a highly relevant distant island can out-compete multiple logits from a closer but much less relevant island.

A simple way to choose which other locations are allowed to compete for the attention of location $\mathbf{x}$ is to sample, without replacement, from a Gaussian centered at $\mathbf{x}$. Higher level embeddings can sample the same number of other locations but from a larger Gaussian. The sampling could be done only once so it was part of the architecture of the net. Alternatively, lacunae in the sampling could be greatly reduced by sampling independently at each time step.

7 Learning Islands

Let us assume that GLOM is trained to reconstruct at its output the uncorrupted version of an image from which some regions have been removed. This objective should ensure that information about the input is preserved during the forward pass and if the regions are sufficiently large, it should also ensure that identifying familiar objects will be helpful for filling in the missing regions. To encourage islands of near identity, we need to add a regularizer and experience shows that a regularizer that simply encourages similarity between the embeddings of nearby locations can cause the representations to collapse: All the embedding vectors may become very small so that they are all very similar and the reconstruction will then use very large weights to deal with the very small scale. To prevent collapse, contrastive learning [Becker and Hinton, 1992, Paccanaro and Hinton, 2001, van den Oord et al., 2018] uses negative examples and tries to make representations that should agree be close while maintaining

[20] This explains why the first stage of a language model can convert a word like "bank" into a single high-dimensional embedding vector rather than needing separate vectors for the "river" and the "money" senses.

separation between representations which should not agree[21].

Contrastive learning has been applied very successfully to learn representations of image crops [Chen et al., 2020a, Bachman et al., 2019, He et al., 2020, Chen et al., 2020b, Tejankar et al., 2020] It learns to make the representations of two different crops of the same image agree and the representations of two crops from different images disagree. But this is not a sensible thing to do if our aim is to recognize objects. If crop 1 contains objects A and B and crop 2 from the same image contains objects B and C, it does not make sense to demand that the representations of the two crops be the same *at the object level*. It does make sense at the scene level, however. For scenes containing one prominent object, it may be possible to recognize objects using representations that are designed to recognize scenes, but as soon as we distinguish different levels of embedding it becomes clear that it would be better to use a contrastive loss function that encourages very similar representations for two locations at level L only if they belong to the same entity at level L. If they belong to different level L entities their level L embeddings should be significantly different.

From the point of view of a location, at all but the top level it needs to decide which other locations its level L embedding should be similar to. It can then learn to resemble those embeddings and be repelled from the embeddings of locations that belong to different objects in the same or other images. Recent work that uses the similarity of patches along a possible object trajectory to influence whether contrastive learning should try to make them more similar has shown very impressive performance at finding correspondences between patches in video sequences [Jabri et al., 2020].

The obvious solution is to regularize the bottom-up and top-down neural networks by encouraging each of them to predict the consensus opinion. This is the weighted geometric mean of the predictions coming from the top-down and bottom-up networks, the attention-weighted average of the embeddings at nearby locations at the previous time-step the previous state of the embedding. Training the inter-level predictions to agree with the consensus will clearly make the islands found during feed-forward inference be more coherent.

An important question is whether this type of training will necessarily cause collapse if it is not accompanied by training the inter-level predictions to be different for negative examples that use the consensus opinions for unrelated spatial contexts. Using layer or batch normalization should reduce the tendency to collapse but a more important consideration may be the achievability of the goal.

When the positive examples in contrastive learning are used to try to extract very similar representations for different patches of the same image, the goal is generally not achievable and the large residual errors will always be trying to

[21] Maintaining separation is quite different from asking representations that should be separate to be far apart. Once two representations are sufficiently different there is no further pressure to push them even further apart.

make the representations collapse. If, however, an embedding at one location is free to choose which embeddings at other locations it should resemble, the goal can be achieved almost perfectly by learning to form islands of identical vectors and attending almost entirely to other locations that are in the same island. This should greatly reduce the tendency towards collapse and when combined with the deep denoising autoencoder objective function and other recent tricks [Grill et al., 2020, Chen and He, 2020] it may eliminate the need for negative examples.

8 Representing coordinate transformations

When neural networks are used to represent shape, they generally work much better if they represent the details of a shape relative to its intrinsic coordinate frame rather than relative to a frame based on the camera or the world [Taylor et al., 2007, Deng et al., 2020].

Work on the use of neural fields for generating images has established that there are much better ways to represent the location than using two scalars for its x and y coordinates [Sitzmann et al., 2020, Mildenhall et al., 2020]. The product of a delta function at the location with both horizontal and vertical sine and cosine waves of various frequencies works well. A similar representation is used in transformers for the position of a word fragment in a sentence.

The success of highly redundant representations of location suggests that there may also be highly redundant representations of the non-translational degrees of freedom of a coordinate transform that work much better in a neural net than the matrices or quaternions commonly used in computer graphics[22]. Let us suppose that we would like the pose of a part (*i.e.* the coordinate transform between the retina and the intrinsic frame of reference of a part) to be represented by a vector that is a subsection of the embedding vector representing the part. A multi-layer neural network whose weights capture the viewpoint-invariant coordinate transform between a part and a whole can then operate on the pose vector of the part to predict the pose vector of the whole. If we simply flatten the 4x4 matrix representation of a pose into a vector, it is easy to hand-design a multi-layer neural net that takes this vector as input and produces as output a vector that corresponds to the flattened result of a matrix-matrix multiply, *provided we know what matrix to multiply by, which depends on the identity of the part.* This dependence on the part identity was the reason for allocating a separate capsule to each type of part in earlier capsule models. Unfortunately, the vector space of flattened 4x4 matrices does not make it easy to represent uncertainty about some aspects of the pose and certainty about others. This may require a much higher-dimensional representation of pose.

[22]The standard matrix representation uses the scale of the matrix to represent the change in scale caused by the coordinate transform. Using the scale of the weights to represent scale in this analog way is a particularly bad idea for neural nets.

Designing this representation by hand is probably inferior to using end-to-end learning with stochastic gradient descent. Nevertheless, section 9 discusses one approach to representing uncertainty in a neural net, just to demonstrate that it is not a major problem.

In a universal capsule the part-identity is represented by an activity vector rather than by the choice of which capsule to activate, so the neural net that implements the appropriate part-whole coordinate transform needs to condition its weights on the part-identity vector [23]. Consequently, the entire part-level vector of a location needs to be provided as input to the bottom-up neural net that computes the part-whole coordinate transform. This makes the computation much more complicated but it greatly simplifies the design of the architecture. It means that we do not need to designate one part of the embedding vector at a level to represent the pose and the rest to represent other aspects of the entity at that level. All we need to do is to make sure that the neural net that predicts the embedding at one level from the embedding below (or above) has sufficient expressive power to apply a coordinate transform to those components of the embedding vector that represent pose and to make this coordinate transform be contingent on those components of the vector that represent the identity of the part. Since this neural net is going to be learned by stochastic gradient descent, we do not even need to keep components of the embedding vector that represent the pose separate from the components that represent other properties of the entity at that level: individual components can be tuned to combinations of pose, identity, deformation, texture *etc.*

Entangling the representations of identity and pose may seem like a bad idea, but how else can a bottom-up prediction from a diagonal line express the opinion that the whole is either a tilted square or an upright diamond? To express this distribution using activities of basis functions, we need basis functions that are tuned to combinations of identity and pose.

Using a small matrix or quaternion to represent pose makes it easy to model the effects of viewpoint changes using *linear* operations. At first sight, abandoning these explicit representations of pose seems likely to compromise the ability of a capsule to generalize across viewpoints. This would be true if each capsule only dealt with one type of object, but universal capsules will have seen many different types of object from many different viewpoints and any new type of object will be well approximated by a weighted average of familiar types all of which have learned to model the effects of viewpoint. Moreover, the weights in this average will be the same for all viewpoints. So if a novel object is only seen from a single viewpoint, a universal capsule may well be able to recognize it from radically different viewpoints.

The same argument about generalization can be applied to CNNs, but there

[23] In stacked capsule autoencoders [Kosiorek et al., 2019] the capsule identity determines the default object-part coordinate transform, but the transform can be modulated by a vector that represents the deformation of the object.

is a subtle difference: GLOM is forced to model the coordinate transforms between parts and wholes correctly in order to be able to make use of the spatial relationship between one part and another by using a simple averaging operation at the level of the whole. It is the viewpoint invariance of these part-whole spatial relationships that makes it possible to generalize to radically new viewpoints.

9 Representing uncertainty

It is tempting to imagine that the individual components of an embedding vector correspond to meaningful variables such as the six degrees of freedom of the pose of an object relative to the camera or the class of an object. This would make it easy to understand the representation, but there is a good reason for making the relationship between physically meaningful variables and neural activities a little less direct: To combine multiple sources of information correctly it is essential to take the uncertainty of each source into account.

Suppose we want to represent M-dimensional entities in such a way that different sources of information can contribute *probability distributions* over the M-dimensional space rather than just point estimates. We could use a population of $N \gg M$ neurons each of which is tuned to a Gaussian in the M-dimensional space [Williams and Agakov, 2002]. If we take logs, a neuron then corresponds to a parabolic bump in the log probability. This bump could be very wide in some directions and very narrow in others. It could even be a horizontal ridge that is infinitely wide in some of the directions. We treat the activity of a neuron as a vertical scaling of its parabolic bump and simply add up all the scaled bumps to get a parabolic bump which represents the log of the unnormalized Gaussian distribution represented by the population of N neurons.

Sources of information can now contribute probability distributions which will be multiplied together by simply contributing additively to the activities of the N neurons. If we want to keep N relatively small, there will be limitations on the probability distributions that can be represented, but, given a budget of N neurons, learning should be able to make good use of them to approximate the predictive distributions that are justified by the data. If, for example, it is possible for a part to predict the horizontal location of a whole without making this prediction be contingent on other aspects of the pose or identity of the whole, it would be helpful to tune a handful of the N neurons to well-spaced values on the dimension representing the horizontal location of the whole in the underlying M-dimensional space. The part can then contribute a Gaussian distribution along this horizontal dimension by making appropriate contributions to this handful of neurons. The relative magnitudes of the contributions determine the mean of the Gaussian and their overall scale determines the inverse variance of the Gaussian.

The assumption that the neurons have Gaussian tuning in the underlying M-dimensional space of possible entities was just a simplification to show that neural networks have no problem in representing Gaussian probability distributions and combining them appropriately. A much more flexible way to tune the neurons would be to use a mixture of a Gaussian and a uniform [Hinton, 2002]. The log of this distribution is a *localized* bump which will be called a unibump. The sides of a unibump splay out and eventually become horizontal when we are far enough from the mean that the uniform completely dominates the Gaussian. Unlike a parabolic bump which has a quadratically large gradient far from its maximum, a unibump has zero gradient far from its maximum so it makes no contribution to the shape of the unnormalized distribution far from its mean. This allows unibumps to represent multi-modal probability distributions. The sum of one set of nearby unibumps can represent one mode and the sum of another set of unibumps that are close to one another but far from the first set can represent another mode. Using neural activities that correspond to vertical scalings of the unibumps, it is possible to control both the location and the sharpness of each mode separately.

The assumption that individual neurons are tuned to a mixture of a Gaussian and a uniform was just a simplification to show that neural networks can represent multi-modal distributions. The basis functions that neurons actually learn for representing multi-modal log probability distributions in an underlying latent space do not need to be local in that space.

The need to represent uncertainty prevents the simplest kind of representation in which activity in a single neuron represents one dimension of an M-dimensional entity, but it still allows neurons to have tuning curves in the M-dimensional space. Whether it is possible for someone trying to understand the representations to jointly infer both the underlying latent space and the tuning curves of the neurons in that space is a very interesting open problem. But even when it is hard to figure out what the individual neurons are doing it should still be trivial to find islands of nearly identical vectors, so it should be easy to see how GLOM is parsing an image or how a similar model applied to language is parsing a sentence.

When considering how to represent uncertainty about the pose or identity of a part, it is very important to realize that each location assumes that it is only occupied by at most one part at each level of the hierarchy[24]. This means that all the neural activities in the embedding vector at a level refer to the same part: there is no binding problem because the binding is done via the location. So a location can use two different neurons whose tuning curves in the underlying M-dimensional space overlap by a lot without causing any confusion. If we do not start by allocating different subsets of the neurons to different locations, the broad tuning curves in the M-dimensional underlying

[24]We assume the visual world is opaque. Transparency, like reflections in a window of an indoor scene superimposed on an outdoor scene would need to be handled by switching attention between the two different scenes.

space that are needed for representing uncertainty will cause confusion between the properties of different objects. That is why coarse coding, which uses a single population of broadly tuned neurons to model several different entities at the same time [Hinton, 1981a] cannot model uncertainty efficiently.

9.1 Combining different sources of information when updating the embeddings

The embedding at each level is updated using information from the previous time-step at adjacent levels and also at other locations on the same level. These sources are far from independent, especially when the image is static so that the visual input is identical at multiple time-steps. The higher-level embeddings obviously depend on the earlier lower-level embeddings. Also, the same-level embeddings that contribute to the attention-weighted local average will have been influenced by early states of the very embedding that the attention-weighted average is trying to update. To avoid becoming over-confident it is better to treat the different sources of information as *alternative* paths for computing the embedding vector from the visual input. This justifies taking a weighted geometric mean of the distributions[25] predicted by the individual sources rather than a simple product of these distributions which would be appropriate if they were independent. For interpreting a static image with no temporal context, the weights used for this weighted geometric mean need to change during the iterations that occur after a new fixation. Initially the bottom-up source should be by far the most reliable, but later on, the top-down and lateral sources will improve. Experiments with deep belief nets[Hinton, 2006] show that gradually increasing the weighting of top-down relative to bottom-up leads to more plausible reconstructions at later times, suggesting that this will be important when GLOM is trained as an end-to-end deep denoising autoencoder.

10 Comparisons with other neural net models

This section compares GLOM to some of the neural net models that influenced its design.

[25] When taking the geometric mean of some distributions we assume that the product of the distributions is renormalized to have a probability mass of 1.

10.1 Comparison with capsule models

The main advantage of GLOM, compared with capsule models [26], is that it avoids the need to pre-allocate neurons to a discrete set of possible parts at each level. The identity of a part becomes a vector in a continuous space of feature activities. This allows for much more sharing of knowledge between similar parts, like arms and legs, and much more flexibility in the number and type of parts belonging to an object of a particular type.

A second advantage of GLOM is that it does not require dynamic routing. Instead of routing information from a part capsule to a specific capsule that contains knowledge about the relevant type of whole, every location that the part occupies *constructs* its own vector representation of the whole. The constraint that a part at one location only belongs to one whole is a necessary consequence of the the fact that the alternative wholes at that location are alternative activity vectors on the same set of neurons. Uncertainty about which of several wholes is the correct parent of a part can still be captured by using blends of activity vectors.

A third advantage of GLOM is that the cluster formation procedure for forming islands is much better than the clustering procedure used in capsule models. To make methods like EM work well when the number of clusters is unknown, it is helpful to introduce split and merge operations [Ueda et al., 2000] but these operations happen automatically during island formation. Hierarchical Bayesian concerns about finding the correct number of clusters at an embedding level are addressed by starting with one island per location and then reducing the number of distinctly different islands by making embedding vectors agree. This reduction occurs in a continuous space with no need for discrete changes in the number of clusters.

The main disadvantage of GLOM, compared to most capsule models, is that knowledge about the shape of a specific type of object is not localized to a small group of neurons (possibly replicated across quite large regions). Instead, the bottom-up and top-down neural nets (which may be different for every pair of adjacent levels) have to be replicated at every single location. For computer implementations the replication across locations is a big advantage because it allows a weight to be used many times each time it is retrieved from memory, but for biological neural nets it seems very wasteful of synapses. The point of the analogy with genes is that biology can afford to be wasteful so this objection may not be as serious as it seems. There is, however, a more serious issue for a biological version of GLOM: The ubiquitous universal capsules would need to learn the very same knowledge separately at every different location and this is *statistically* very inefficient. Fortunately, section 12 shows how locations can share what their bottom-up and top-down models have learned without sharing

[26] Some capsule models already use universal capsules in which vectors of activity rather than groups of neurons are used to represent the part identity, but they do not replicate these vectors across all locations within the object [Srivastava et al., 2019].

any of their weights.

By allocating neurons to locations rather than to types of object or part, GLOM eliminates a major weakness of capsule models, but it preserves most of the good aspects of those models:

- **Handling the effects of viewpoint properly:** The weights of the bottom-up and top-down neural networks capture the viewpoint-invariant spatial relationships between parts and wholes and the neural activities capture the viewpoint equivariant information about the pose of an object or part.
- **Coincidence filtering:** Objects are recognized by using agreement between high-dimensional predictions from their parts. In GLOM, the idea of using agreement is taken even further because it is also used to *represent* objects and parts as islands of identity.
- **No dynamic allocation of neurons:** The part-whole hierarchy can be represented without dynamically allocating neurons to nodes in the parse tree.

10.2 Comparison with transformer models

The GLOM architecture shown in figure 1 can be rearranged by viewing the vertical time-slices in figure 1 as layers in figure 4. This rearrangement of GLOM is then equivalent to a standard version of a transformer [Vaswani et al., 2017] but with the following changes:

- The weights are the same at every layer because GLOM is a recurrent net and we have converted the time slices into layers.
- The attention mechanism is greatly simplified by using the embedding vector at a level as the query, the key and also the value. The complex interactions between different locations that are normally implemented by attention are thus reduced to a simple, attention-weighted, smoothing operation.
- The multiple heads used to provide more expressive power in most transformers are re-purposed to implement the multiple levels of a part-whole hierarchy and the interactions between the heads at a location are highly structured so that a level only interacts with the adjacent levels.
- The bottom-up and top-down neural networks that compute the interactions between adjacent levels perform coordinate transformations between the distributed representations of the poses of parts and wholes and these coordinate transformations depend on the distributed representations of the types of the part and the whole.

The justification for eliminating the distinction between the query, the key, the value and the embedding itself is as follows: Consider trying to get a potential mouth to be corroborated by a potential nose in a transformer. The mouth needs to ask "is there anyone in the right spatial relationship to me who could be a nose". If so, please tell me to be more mouth-like. This seems to require the mouth to send out a nose query (that includes the appropriate pose relative to the mouth) that will match the key of the nose. The nose then needs to send back a mouth-like value (that includes the appropriate pose relative to the nose).

But the mouth could also be corroborated by an eye so it needs to send out a different query that will match the key of an eye. This could be handled by using separate heads for a mouth-looking-for-a-nose and a mouth-looking-for-an-eye (as in categorial grammar), but that seems clumsy.

A more elegant solution (inherited from capsule models) is to use a form of the Hough transform. The potential mouth predicts a vector for the face it might be part of. The potential nose and eye do the same. All you need now is agreement of the predictions at the face level so query=key=value=embedding. The face level can then give top-down support to its parts instead of the support coming from a value vector sent by one part to another using a coordinate transform specific to the identities of the two parts.

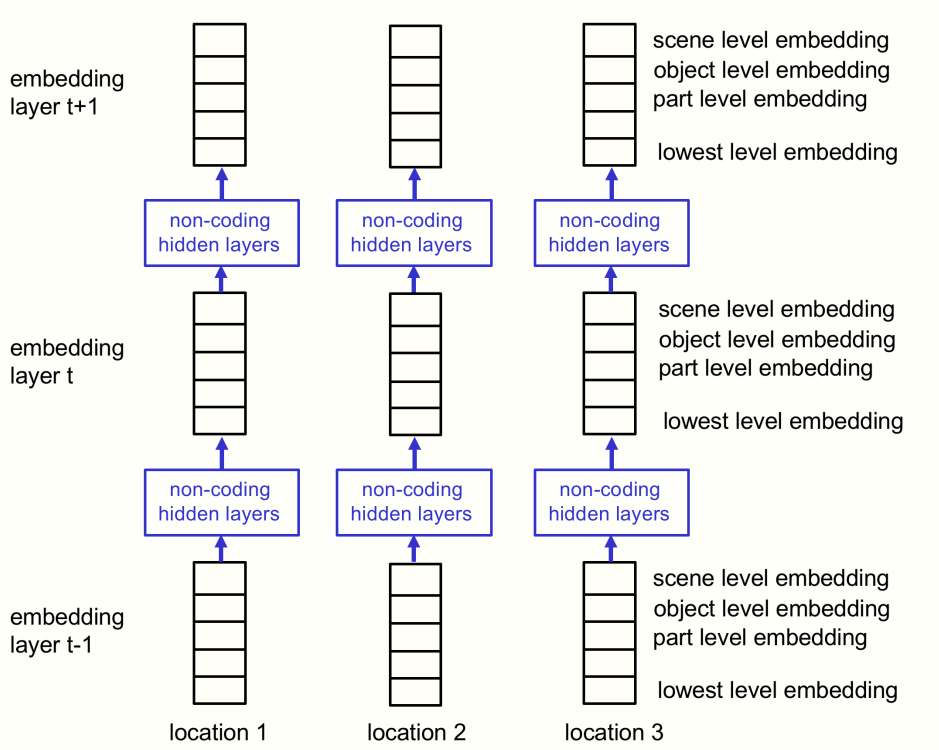


Figure 4: This is a different way of visualizing the architecture shown in figure 1 which makes the relationship of that architecture to transformers more obvious. The horizontal dimension which represents time in figure 1 becomes the vertical dimension which represents layers in this figure. At each location, every layer now has embeddings for all of the levels in the part-whole hierarchy. This corresponds to vertically compressing the depiction of the levels within a single time-slice in figure 1. A single forward pass through this architecture is all that is required to interpret a static image. All of the level-specific bottom-up and top-down neural nets are shown here as a single neural net. Figure 5 shows the individual bottom up and top-down neural nets for this alternative way of viewing the GLOM architecture.

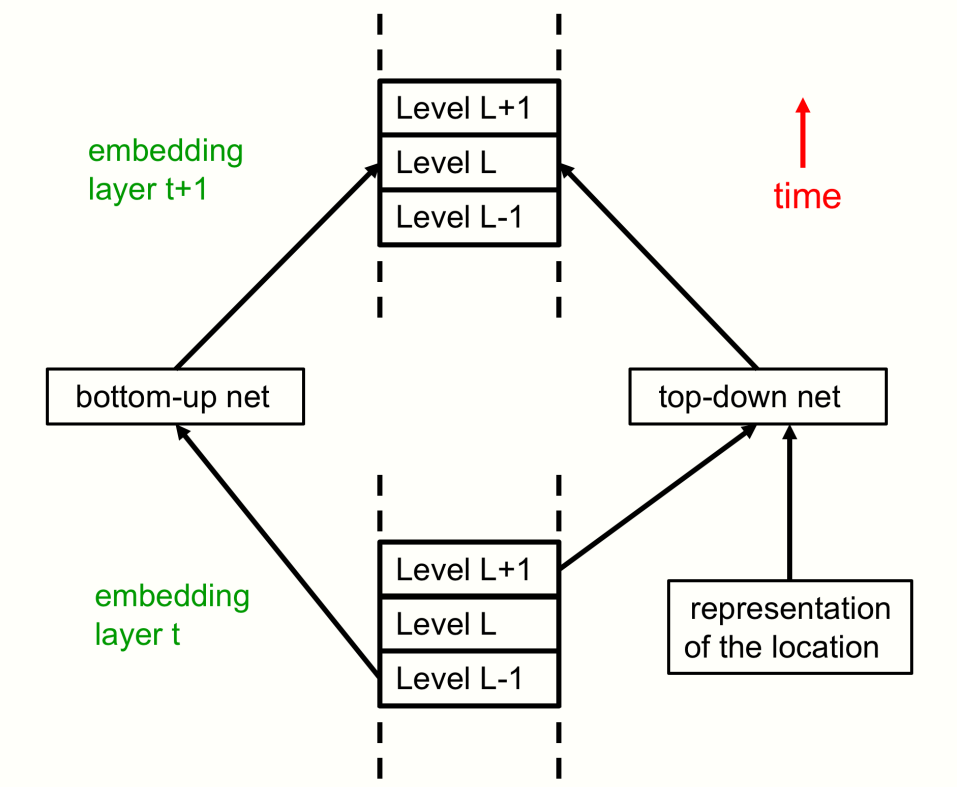


Figure 5: A picture of two adjacent layers of GLOM for a single location (*i. e.* part of a single column). During the forward pass, the embedding vector at level L receives input from the level L-1 embedding vector in the previous layer via a multi-layer bottom-up neural net. It also receives input from the level L+1 embedding in the previous layer via a multi-layer top-down neural net. The dependence on level L+1 in the previous layer implements top-down effects during the forward pass. The level L embedding in layer $t + 1$ also depends on the level L embedding in layer t and an attention-weighted sum of the level L embeddings at other nearby locations in layer t. These within-level interactions are not shown.

10.3 Comparison with convolutional neural networks

Capsules were originally motivated by three perceived deficiencies of CNNs:

1. The pooling operation in a CNN was designed to achieve local *invariance* to translation in the activity vector at the next level up. It seems better to ask for invariance in the weights but equivariance in the activities.

2. CNNs attempt to generalize across viewpoints by using a lot of examples of different viewpoints which may be produced by augmenting the dataset with transformed images. Computer graphics generalizes across viewpoints by having explicit representations of the poses of objects or parts relative to the camera. A change in viewpoint, even a very big one, can be modeled perfectly by *linear* operations on these explicit poses. Using the viewpoint invariant relationship between the pose of a part and the pose of the whole seems like a very efficient way to generalize to radically new viewpoints. CNNs do not appear to be doing this, but appearances can be deceptive.

3. In CNNs, the activity of a neuron is determined by the scalar product of a weight vector with an activity vector. This is not a good way to model covariance structure which is very important in vision. Taking the scalar product of an activity vector with another activity vector makes powerful operations like coincidence detection and attention much easier. Coincidences in a high-dimensional embedding space are a good way to filter out noise caused by occlusion or missing parts because, unlike sums, they are very robust to the absence of some of the coinciding predictions.

The **first** deficiency is only apparent. It depends on a common misunderstanding of how CNNs represent the positions of parts. The vector of channel activities at a gridpoint can have a rate-coded representation of the position of a part that is far more accurate than the stride between gridpoints. So when the stride is increased at the next level by pooling it does *not* mean that the position of a part is encoded less accurately. A bigger stride does not produce representations that are more translationally invariant. Gridpoints are used to allocate neural hardware not to represent positions. Their spacing is limited by the fact that the neural hardware at a gridpoint assumes it will never be representing more than one of whatever it represents, not by the accuracy with which position needs to be represented.

Attempts to deal with the **second** perceived deficiency led to some interesting models. The transforming autoencoder [Hinton et al., 2011] forced an encoder to extract an explicit representation of pose in each capsule by insisting that the reconstructed image should be a transformed version of the original image and specifying this transformation as a matrix that multiplied whatever matrix elements were extracted by the encoder. Similarly, the EM capsule model

extrapolated much better to new viewpoints when it was forced to use a matrix to represent the relationship of a part to a whole.

Unfortunately, perception has to deal with uncertainties that are not present in computer graphics[27] and it needs to be able to represent correlated uncertainties in its pose predictions so that multiple sources of information can be combined properly. This rules out a simple matrix representation of pose. Once we accept that distributions over the possible poses of an entity will be represented by the scales assigned to basis functions in the log probability space, it is quite possible that CNNs actually learn to do something like this. This might allow them to approximate Hough transforms, though this is hard to do without taking scalar products of activity vectors.

The **third** deficiency can be rectified by moving to a transformer-like architecture that uses scalar products of activity vectors to modulate attention.

If you like CNNs, GLOM can be viewed as a special type of CNN that differs from a standard CNN in the following ways:

- It only uses 1x1 convolutions (except at the front end).
- Interactions between locations are done by parameter-free averaging that implements a coincidence filter which allows it to use a Hough transform to activate units rather than only using matched filters.
- Rather than using a single feed-forward pass through the levels of representation, it iterates to allow top-down influences that are implemented by neural fields.
- It includes contrastive self-supervised learning and performs hierarchical segmentation as a part of recognition rather than as a separate task. No more boxes.

10.4 Representing the ISA hierarchy

An important idea in Good Old-Fashioned Artificial Intelligence (GOFAI) is property inheritance. It is not necessary to explicitly represent that elephants suckle their young because an elephant ISA mammal and, unless otherwise stated, an elephant inherits this property from its more general type. A simple way to implement property inheritance in a neural network is to make different entities correspond to different vectors of activity on the same set of neurons.

[27] Even if a generative model is stochastic, it may still be certain about which stochastic choices it made. In some more complex generative models, however, a level only specifies the probability distributions of poses for parts at the level below and an iterative process then reconciles these distributions [Osindero and Hinton, 2008]. This kind of generative model may be needed for modelling very precise relationships between highly variable parts, such as two praying hands, and it *does* need to be able to represent probability distributions over poses.

Imagine that the components of the vector that represents a concept are ordered from very general to very specific. Mammals all have similar values for the more general components and differ on less general components. Indian and African elephants only differ on fairly specific components. When a neural net learns to make the vectors for concepts have causal effects on other vectors, effects which should be the same for all mammals but not the same for all vertebrates will naturally be implemented by the outgoing weights of the neurons that are active for all mammals but not for all vertebrates. This way of implementing property inheritance makes it easy to add exceptions. The components of a vector that are common to birds will learn weights that capture the knowledge that birds fly and the more specific components that differentiate penguins from other birds will learn stronger weights that overrule the general case [Hinton, 1981b].

This way of implementing property inheritance has the added advantage that types do not need to form a tree. Dogs inherit many properties from being canines (like wolves) but they also inherit many properties from being pets (like cats). There is no guarantee that properties inherited from these more general, partially overlapping classes will be consistent, but, unlike logic, neural networks have no difficulty dealing with conflicting evidence.

At first sight, the idea of using different sections of the vector representation of a concept to capture different levels in the ISA hierarchy conflicts with the idea of using different sections to capture different levels in the part-whole hierarchy. This seems problematic because *hooked beak* is a part of a bird but it also defines a type of bird. The two ideas can be reconciled by first dividing the embedding vector for a location into sections that represent different levels in the part-whole hierarchy and then dividing each section into subsections that represent different levels in the type hierarchy.

10.5 The relationship to 2-D Ising models

For each location separately, the embedding vectors at levels L-1 and L+1 on the previous time-step provide input to the neurons that represent the current embedding vector at level L. This acts like the conditioning input in a conditional Markov Random Field: it influences the current step of the iterative, island forming process that tries to make the embedding of the location at level L agree with the embeddings of other locations at level L.

In a 2-D Ising model, a two-dimensional array of binary-valued spins settles into a state in which nearby spins tend to agree so as to minimize an energy function that favors agreement between neighboring spins. The model proposed here resembles the 2-D Ising model because it uses a 2-D grid of image locations but it generalizes the model in the following ways:

1. It replaces binary spins with high-dimensional real-valued vectors. The fact that these lie in a continuous space should facilitate the search for

islands of agreement.

2. it replaces a single field of spins with fields at multiple levels, and allows adjacent level embeddings of the same location to interact [He et al., 2004, Saremi and Sejnowski, 2013]. The interactions between levels are quite complicated because they involve coordinate transformations between parts and wholes. So for each pair of adjacent embedding levels, the top-down and bottom-up interactions at each location must be computed by a multilayer neural net rather than a simple weight matrix.

10.6 Comparison with other methods for removing redundancy

Methods like principal components analysis remove redundancy in the data by limiting the number of available dimensions in the representation. By contrast, a restricted Boltzmann machine with a large number of hidden units squeezes out redundancy by making nearly all of the exponentially many possible binary configurations of the hidden units have such high energy that they are effectively unavailable. This is a much more flexible way of eliminating redundancy [Shi and Zhu, 2007]. It can model multiple fat manifolds[28] that have different intrinsic dimensionalities and even within a fat manifold it can model variations in the effective dimensionality in different parts of the manifold. The island forming objective belongs to the second class of methods. At each level, it allows for a large number of small islands if that is what the data requires but strives to use a small number of large islands if that is possible.

11 Video

This paper focuses on using the GLOM architecture to process a single fixation of a static image, but the architecture is motivated by the need to deal with video and learning from video is often much easier than learning from static images [Sabour et al., 2021], so I will briefly discuss the simplest temporal extension which is to a single fixation of a time-varying image.

To avoid confusion it may be helpful to distinguish three different types of time:

- **Event time:** This is the actual time at which an event occurs.

[28] A manifold is a subset of the points in a space that have lower intrinsic dimensionality than the full space. If we take the points on a manifold and add a small amount of noise that has full dimensionality, the points no longer form a strict manifold, but they will all be close to the manifold. Such a set of points will be said to lie on a fat manifold.

- **Representation time:** This is the actual time at which a particular representation of an event occurs in the neural network. If the bottom-up neural network uses a predictive model, representations of events could be in synchrony with the events themselves or they could even precede the events which would make catching a ball a lot easier.
- **Reference time:** This is the actual time that an internal representation refers to. When a memory is retrieved, for example, the reference time of the constructed representation is usually long before the representation time. The reference time can also differ by a lot from the event time if the memory is not veridical.

For a sequence of frames representing a static image, multiple time steps can be used to settle on an appropriate set of islands at each level. But in a dynamic image, the very same time-steps must also be used to deal with the fact that the occupants of a location at each level can change with time.

An advantage of using islands of identical vectors to represent an object is that motions between successive frames that are small compared to the size of the object only require large changes to a small subset of the locations at the object level. All of the locations that remain within the object need to change only sightly to represent the slight change in pose of the object relative to the camera.

If the changes in an image are small and predictable, the time-steps immediately following a change of fixation point can be used to allow the embeddings at all levels to settle on slowly changing islands of agreement that track the changes in the dynamic image. The lowest level embeddings may change quite rapidly but they should receive good top-down predictions from the more stable embeddings at the level above. Once the embeddings have formed sensible islands, there is then no problem in using the very same time-step for improving the interpretation of each frame and for keeping the embeddings locked on to the dynamic image.

If the changes are rapid, there is no time available to iteratively settle on a good set of embedding vectors for interpreting a specific frame. This means that the GLOM architecture cannot correctly interpret complicated shapes if the images are changing rapidly. Try taking a irregularly shaped potato and throwing it up in the air in such a way that it rotates at one or two cycles per second. Even if you smoothly track the potato, you cannot see what shape it is.

12 Is GLOM biologically plausible?

Although GLOM is biologically inspired, it has several features that appear to make it very implausible as a biological model. Three of these features are

addressed here.

- The weight-sharing between the bottom-up or top-down models in different columns.
- The need to process negative pairs of examples for contrastive learning without interrupting the video pipeline.
- The use of backpropagation to learn the hidden layers of the top-down and bottom-up models.

12.1 Is the neocortex a giant distillery?

The replication of DNA in every cell is unproblematic: that is what DNA is good at. But biologists often object to models that use weight-sharing claiming that there is no obvious way to replicate the weights [Lillicrap et al., 2020]. GLOM, however, suggests a fairly simple way to solve this problem by using contextual supervision. In a real brain, what we want is an efficient way of training the bottom-up and top-down nets at a location so that they compute the same function as the corresponding nets at other locations. There is no need for the weights to be identical as long as corresponding nets are functionally identical. We can achieve this using knowledge distillation [Buciluǎ et al., 2006, Hinton et al., 2014]. For each level separately, the two students at each location are the bottom-up and top-down neural nets. The teacher is the consensus opinion that is a weighted geometric mean of the opinions of the two students, the previous state of the embedding, and the attention-weighted embeddings at *other* locations[29].

Regressing a student's prediction towards the consensus, allows knowledge in the neural nets at other locations to be transferred to the student via the attention weighted averaging. It is not as effective as sharing weights with those other neural nets, but it works quite well [Hinton et al., 2014] and in the long run all of the networks will converge to very similar functions if the data distribution is translation invariant. In the long run, however, we are all dead[30]. So it is interesting to consider what happens long before convergence when the local models are all fairly different.

Suppose all of the locations that form a nose have the same embedding vector at the part level. If they all had exactly the same bottom-up model, they would all make exactly the same prediction for the face at the object level. But if the

[29] Strictly speaking, this is an example of co-distillation where the ensemble of all the students is used as the teacher. Co-distillation was initially based on an analogy with how scientists in a community learn from each other [Hinton, 2014], but the same mechanism could be used in a community of columns. In both cases it would help to explain how a system can win by just replicating a lot of people or columns without requiring any significant architectural innovation.

[30] There are alternative facts.

bottom-up models at different locations are somewhat different, we will get a strong ensemble effect at the object level: The average of all the simultaneous bottom-up predictions for the same object in different locations will be much better than the individual predictions.

One advantage of sharing knowledge between locations via distillation rather than by copying weights is that the inputs to the bottom-up models at different locations do not need to have the same structure. This makes it easy to have a retina whose receptive fields get progressively larger further from the fovea, which is hard to handle using weight-sharing in a convolutional net. Many other aspects, such as the increase in chromatic aberration further from the fovea are also easily handled. Two corresponding nets at different locations should learn to compute the same function of the optic array even though this array is pre-processed differently by the imaging process before being presented to the two nets. Co-distillation also means that the top-down models do not need to receive their location as an input since it is always the same for any given model.

Finally, using distillation to share knowledge between location specific neural networks solves a puzzle about the discrepancy between the number of synapses in the visual system (about 10^{13}) compared to the number of fixations we make in our first ten years (about 10^9). Conservative statisticians, concerned about overfitting, would prefer these numbers to be the other way around[31]. If we use, say, 10^4 columns in different locations, the bottom-up and top-down models at one location only have about 10^9 synapses between them. Conversely, the number of training examples used to learn the knowledge that is shared across an ensemble of 10^4 locations is about 10^{13}, though many of these examples are very highly correlated.

Neural networks that have more training cases than parameters are less magical than some of the highly over-parameterized networks in current use but they may generalize in more predictable ways when presented with data that lies outside their training distribution because the function they compute has been much more highly constrained by the data.

12.2 A role for sleep in contrastive learning?

If negative examples are required, GLOM might appear less plausible as a biological model because of the added complexity of finding and processing pairs of images that are similar when they should not be. There is, however, one intriguing possibility that emerged from conversations with Terry Sejnowski in 1983 and 2020.

When using contrastive learning to get representations that are similar for neighbouring video frames, the most effective negative examples are frames in

[31] It should help that each example allows contrastive learning at several different levels and the target vectors for contrastive learning are much richer than a single one-of-N label

the same video that are nearby but not immediately adjacent. We could avoid compromising the real-time performance of GLOM by taking it offline at night to do the negative learning that prevents the representations from collapsing. If the highest level embeddings have the ability to generate sequences at the highest level, the top-down networks could be used to generate sequences of embeddings at every level in each column. This process does not require any attention between columns because it does not need to perform perceptual inference, so it might be able to generate plausible sequences at a much faster rate than the normal speed. Then we simply do the negative learning for the bottom-up models using the same length of real-time window as is used when awake. There is evidence that high-speed, top-down sequence generation occurs during the spindle stage of sleep [Lee and Wilson, 2002, Nádasdy et al., 1999].

The idea that sleep is used to keep apart representations that should not be confused is not new [Crick and Mitchison, 1983]. Hinton and Sejnowski [Hinton and Sejnowski, 1986] even suggested that sleep could be used for following the derivatives of the normalizing term in the negative phase of Boltzmann machine learning. But this reincarnation of the idea has two big advantages over Boltzmann machines. First, contrastive unsupervised learning scales much better than Boltzmann machine learning and second, it is far more tolerant of a temporal separation between the positive and negative phases.

Preliminary experiments using contrastive learning for MNIST digits show that the learning still works if a large number of positive updates are followed by a large number of negative updates. Representation collapse is fairly slow during the positive-only learning and the representations can shrink by a significant factor without much affecting performance. So maybe some pairs of embeddings that ought to be well separated get too close together during the day and are then pushed apart again at night. This would explain why complete sleep deprivation for a few days causes such serious mental confusion[32]. The experiments with MNIST also show that after a lot of positive-only learning, performance degrades but is rapidly restored by a small amount of negative learning.

To avoid very long periods of negative-only learning, it might be advisable to start with a negative phase of sleep to push representations apart, and then alternate with a positive phase using input sequences that were generated from the top level or even from a recurrent network close to the sensory input. This conflicts with the Crick-Mitchison theory that REM sleep is for unlearning, but it would still be compatible with our failure to remember almost all of our dreams if episodic memory retrieval depends on the top-level, and the top-level simply does not learn during REM sleep because those episodes simply did not happen.

[32] If sleep is just for doing extra rehearsal or for integrating the day's experiences with older experiences, it is not clear why complete lack of sleep for a few days has such devastating effects.

12.3 Communicating error derivatives in the brain

The straightforward way to train GLOM is to ask it to fill in missing regions of images and to backpropagate the reconstruction error through the entire temporal settling process using backpropagation through time. The contrastive representation learning at each level can then be viewed as an additional regularizer. Unfortunately, it is hard to see how a brain could backpropagate through multiple time steps. If, however, the consensus opinion at every level can provide a sufficient teaching signal for the bottom-up and top-down models that predict the embedding vector at that level, implementation in a brain becomes a lot more feasible.

If we could ensure that the representations improved over time, the temporal derivatives of neural activities could represent error derivatives and the local learning procedure would then be spike-time dependent plasticity in which the increase in a synapse strength is proportional to the product of the pre-synaptic activity with the post-synaptic rate of increase of activity.[33] Assuming spikes are caused by an underlying rate variable, we can get a noisy but unbiased estimate[34] of the rate of change of this underlying rate variable by applying a derivative filter to the post-synaptic spike train, which is exactly what STDP does.

A recent review paper [Lillicrap et al., 2020] discusses at great length how temporal derivatives can be used as error derivatives in order to approximate backpropagation in a feedforward network[35]. The review paper assumes a separate phase in which derivatives, in the form of activity perturbations, are allowed to flow back from the higher levels to the lower levels. This process does not seem plausible for a video pipeline. By contrast, the settling process of GLOM propagates the derivatives required for learning as the temporal derivatives of activity at all levels and the time steps required for this propagation can be the very same time steps as are used for video frames.

For dynamic images, it may seem paradoxical that the representations just keep getting better, but it is no more paradoxical than a surfer who just keeps going downhill without ever changing her elevation. The surface on which the surfer is going downhill is not the same surface. Similarly the time-slice of reality for which the representations are forever improving is not the same time slice. The brain surfs reality.

Unfortunately, this does not explain how to get the derivatives required for

[33] This fits very well with the strongly held beliefs of Jeff Hawkins and others that the brain learns by predicting what comes next.

[34] Stochastic gradient descent is extremely tolerant of noise in the gradient estimates so long as they are unbiased.

[35] This paper summarizes the results of simulations that show that this proposal can be made to work quite well, but not as well as vanilla CNNs. A significant contributor to the performance gap is the statistical inefficiency caused by the lack of weight-sharing and co-distillation should help to fix this.

learning the hidden layers of the bottom-up and top-down neural networks. Nor does it explain how the derivatives of the error signals at each level are backpropagated through the bottom-up or top-down networks to make the appropriate contributions to the derivatives at adjacent levels. Those thorny issues are addressed in another paper which is in preparation.

13 Discussion

This paper started life as a design document for an implementation but it was quickly hijacked by the need to justify the design decisions. I have used the imaginary GLOM architecture as a vehicle for conveying a set of interconnected ideas about how a neural network vision system might be organized. The absence of a working implementation makes it easier to focus on expressing the ideas clearly and it avoids the problem of confounding the quality of the ideas with the quality of the implementation, but it also creates serious credibility concerns. The difference between science and philosophy is that experiments can show that extremely plausible ideas are just wrong and extremely implausible ones, like learning a entire complicated system by end-to-end gradient decent, are just right. I am currently collaborating on a project to test out the ability of the GLOM architecture to generalize shape recognition to radically new viewpoints and I am hoping that other groups will also test out the ideas presented here. This paper has gone on long enough already so I will conclude by making some brief philosophical comments.

The idea that nodes in a parse tree are represented by islands of similar vectors unifies two very different approaches to understanding perception. The first approach is classical Gestalt psychology which tried to model perception by appealing to fields and was obsessed by the idea that the whole is different from the sum of the parts[36]. In GLOM, a percept really is a field and the shared embedding vector that represents a whole really is very different from the shared embedding vectors that represent the parts. The second approach is classical Artificial Intelligence which models perception by appealing to structural descriptions. GLOM really does have structural descriptions, and each node in the parse tree has its own "address" but the addresses live in the continuous space of possible embeddings, not in the discrete space of hardware locations.

Some critics of deep learning argue that neural nets cannot deal with compositional hierarchies and that there needs to be a "neurosymbolic" interface which allows neural network front- and back-ends to hand over the higher-level reasoning to a more symbolic system[37]. I believe that our primary mode of reasoning is by using analogies which are made possible by the similarities between learned high-dimensional vectors, and a good analogy for the neurosymbolic in-

[36] Thanks to George Mandler for this more accurate translation.

[37] This is reminiscent of Cartesian dualism which postulated an interface between the body and the mind.

terface is a car manufacturer who spends fifty years expounding the deficiencies of electric motors but is eventually willing to use them to inject the gasoline into the engine.

The phenomenal success of BERT [Devlin et al., 2018], combined with earlier work that demonstrates that neural networks can output parse trees if that is what the task requires [Vinyals et al., 2014], clearly demonstrates that neural networks can parse sentences if they want to. By structuring the interactions between the multiple heads in BERT so that they correspond to levels of representation and by adding a contrastively learned regularizer to encourage local islands of agreement over multiple word fragments at each level, it may be possible to show that GLOMBERT actually does parse sentences.

Acknowledgments

Many people helped me arrive at the set of ideas described in this paper. Terry Sejnowski, Ilya Sutskever, Andrea Tagliasacchi, Jay McClelland, Chris Williams, Rich Zemel, Sue Becker, Ruslan Salakhutdinov, Nitish Srivastava, Tijmen Tieleman, Taco Cohen, Vincent Sitzmann, Adam Kosoriek, Sara Sabour, Simon Kornblith, Ting Chen, Boyang Deng and Lala Li were particularly helpful. People who helped me to improve the presentation of the ideas include David Fleet, David Ha, Michael Isard, Keith Oatley, Simon Kornblith, Lawrence Saul, Tim Shallice, Jon Shlens, Andrea Tagliasacchi, Ashish Vaswani and several others. I would especially like to thank Jeff Dean and David Fleet for creating the environment at Google that made this research possible. There are probably many highly relevant papers that I should have read but didn't and I look forward to learning about them.

References

[Ba et al., 2016] Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016). Using fast weights to attend to the recent past. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, pages 4331–4339. Curran Associates, Inc. 11

[Bachman et al., 2019] Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. In Wallach, H., Larochelle, H., Beygelzimer, A., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 15535–15545. Curran Associates, Inc. 17

[Barham and Isard, 2019] Barham, P. and Isard, M. (2019). Machine learning systems are stuck in a rut. In *HotOS '19: Proceedings of the Workshop on Hot Topics in Operating Systems*, page 177–183. 7

[Bear et al., 2020] Bear, D. M., Fan, C., Mrowca, D., Li, Y., Alter, S., Nayebi, A., Schwartz, J., Fei-Fei, L., Wu, J., Tenenbaum, J. B., and Yamins, D. L. K. (2020). Learning physical graph representations from visual scenes. *arXiv:2006.12373*. 6

[Becker and Hinton, 1993] Becker, S. and Hinton, G. (1993). Learning mixture models of spatial coherence. *Neural Computation*, 5(2):267–277. 13

[Becker and Hinton, 1992] Becker, S. and Hinton, G. E. (1992). A self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355:6356:161–163. 16

[Buciluǎ et al., 2006] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 535–541. 33

[Chen et al., 2020a] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607. 17

[Chen et al., 2020b] Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020b). Big self-supervised models are strong semi-supervised learners. *arXiv:2006.10029*. 17

[Chen and He, 2020] Chen, X. and He, K. (2020). Exploring simple siamese representation learning. *arXiv:2011.10566v1*. 18

[Crick and Mitchison, 1983] Crick, F. and Mitchison, G. (1983). The function of dream sleep. *Nature*, 304:111–114. 35

[Deng et al., 2020] Deng, B., Lewis, J. P., Jeruzalski, T., Pons-Moll, G., Hinton, G., Norouzi, M., and Tagliasacchi, A. (2020). NASA: Neural articulated shape approximation. In *Proceedings of the European Conference on Computer Vision*. 18

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*. 2, 3, 38

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv:2010.11929*. 12

[Geman and Geman, 1984] Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741. 13

[Grill et al., 2020] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning. *arXiv:2006.07733*. 18

[Ha, 2016] Ha, D. (2016). Generating large images from latent vectors. *blog.otoro.net*. 7, 9

[He et al., 2020] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 17

[He et al., 2004] He, X., Zemel, R. S., and Carreira-Perpinan, M. (2004). Multiscale conditional random fields for image labeling. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 31

[Hinton, 1979] Hinton, G. (1979). Some demonstrations of the effects of structural descriptions in mental imagery. *Cognitive Science*, 3(3):231–250. 1, 10

[Hinton, 1981a] Hinton, G. (1981a). Shape representation in parallel systems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence Vol 2*, pages 1088–1096. 22

[Hinton, 1990] Hinton, G. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75. 11

[Hinton, 2006] Hinton, G. (2006). Grant proposal to the natural sciences and engineering research council. *https://www.cs.toronto.edu/ hinton/absps/NSERC06.pdf*. 22

[Hinton, 2014] Hinton, G. (2014). Dark knowledge. *Talk at Toyota Technical Institute, Chicago: Available on YouTube*. 33

[Hinton et al., 2011] Hinton, G., Krizhevsky, A., and Wang, S. (2011). Transforming auto-encoders. In Honkela, T., editor, *ICANN 2011: Artificial Neural Networks and Machine Learning*, page 44–51. Springer-Verlag. 28

[Hinton and Sejnowski, 1986] Hinton, G. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge, MA. 35

[Hinton et al., 2014] Hinton, G., Vinyals, O., and Dean, J. (2014). Distilling the knowledge in a neural network. In *NIPS 2014 Deep Learning Workshop*. 33

[Hinton, 1981b] Hinton, G. E. (1981b). Implementing semantic networks in parallel hardware. In Hinton, G. E. and Anderson, J. A., editors, *Parallel Models of Associative Memory*. Lawrence Erlbaum Assoc. 30

[Hinton, 1981c] Hinton, G. E. (1981c). A parallel computation that assigns canonical object-based frames of reference. In *The 7th International Joint Conference on Artificial Intelligence, Volume 2*, page 683–685. Morgan Kaufmann Publishers, Inc. 6

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800. 21

[Hinton et al., 2018] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with EM routing. In *International Conference on Learning Representations*. 2, 7, 14

[Jabri et al., 2020] Jabri, A., Owens, A., and Efros, A. A. (2020). Space-time correspondence as a contrastive random walk. *arXiv:2006.14613*. 17

[Kosiorek et al., 2019] Kosiorek, A., Sabour, S., Teh, Y. W., and Hinton, G. E. (2019). Stacked capsule autoencoders. In Wallach, H., Larochelle, H., Beygelzimer, A., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 15512–15522. Curran Associates, Inc. 2, 7, 14, 19

[Lee and Wilson, 2002] Lee, A. K. and Wilson, M. A. (2002). Memory of sequential experience in the hippocampus during slow wave sleep. *Neuron*, 36(6):1183–1194. 35

[Lee et al., 2019] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 3744–3753. 8, 14

[Lillicrap et al., 2020] Lillicrap, T., Santoro, A., Marris, L., Akerman, C., and Hinton, G. E. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21:335–346. 33, 36

[Locatello et al., 2020] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention. *arXiv:2006.15055*. 7

[Mildenhall et al., 2020] Mildenhall, B., Srinivasan1, P. P., Tancik1, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision ECCV 2020: Computer Vision – ECCV 2020*, pages 405–421. 9, 18

[Nádasdy et al., 1999] Nádasdy, Z., Hirase, H., Czurkó, A., Csicsvari, J., and Buzsáki, G. (1999). Replay and time compression of recurring spike sequences in the hippocampus. *Journal of Neuroscience*, 19(21):9497–9507. 35

[Neal and Hinton, 1997] Neal, R. and Hinton, G. (1997). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I., J., editor, *Learning in Graphical Models*, pages 355–368. Springer, Dordrecht. 8

[Niemeyer and Geiger, 2020] Niemeyer, M. and Geiger, A. (2020). Giraffe: Representing scenes as compositional generative neural feature fields. *arXiv:2011.12100*. 9

[Oore et al., 1997] Oore, S., Hinton, G. E., and Dudek, G. (1997). A mobile robot that learns its place. *Neural Computation*, 9(3):683–699. 9

[Osindero and Hinton, 2008] Osindero, S. and Hinton, G. (2008). Modeling image patches with a directed hierarchy of Markov random fields. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*. 29

[Paccanaro and Hinton, 2001] Paccanaro, A. and Hinton, G. E. (2001). Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244. 16

[Rao and Ballard, 1999] Rao, R. and Ballard, D. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2:79–87. 2

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 3856–3866. Curran Associates, Inc. 2, 7

[Sabour et al., 2021] Sabour, S., Tagliasacchi, A., Yazdani, S., Hinton, G., and Fleet, D. (2021). Unsupervised part representation by flow capsules. *arXiv:2011.13920v2*. 31

[Saremi and Sejnowski, 2013] Saremi, S. and Sejnowski, T. J. (2013). Hierarchical model of natural images and the origin of scale invariance. *Proceedings of the National Academy of Sciences*, 110(8):3071–3076. 31

[Shi and Zhu, 2007] Shi, K. and Zhu, S. (2007). Mapping natural image patches by explicit and implicit manifolds. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. 31

[Sitzmann et al., 2020] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems 33*. 4, 9, 18

[Sitzmann et al., 2019] Sitzmann, V., Zollhoefer, M., and Wetzstein, G. (2019). Scene representation networks: Continuous 3D-structure-aware neural scene representations. In Wallach, H., Larochelle, H., Beygelzimer, A., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32, pages 1121–1132. Curran Associates, Inc. 7

[Srivastava et al., 2019] Srivastava, N., Goh, H., and Salakhutdinov, R. (2019). Geometric capsule autoencoders for 3D point clouds. *arXiv:1912.03310*. 7, 23

[Sun et al., 2020a] Sun, W., Jiang, W., Trulls, E., Tagliasacchi, A., and Yi, K. M. (2020a). ACNe: Attentive context normalization for robust permutation-equivariant learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11286–11295. 13

[Sun et al., 2020b] Sun, W., Tagliasacchi, A., Deng, B., Sabour, S., Yazdani, S., Hinton, G., and Yi, K. M. (2020b). Canonical capsules: Unsupervised capsules in canonical pose. *arXiv:2012.04718*. 7, 10

[Taylor et al., 2007] Taylor, G. W., Hinton, G. E., and Roweis, S. (2007). Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems, 19*. MIT Press. 18

[Tejankar et al., 2020] Tejankar, A., Koohpayegani, S. A., Pillai, V., Favaro, P., and Pirsiavash, H. (2020). ISD: Self-supervised learning by iterative similarity distillation. *arXiv:2012.09259*. 17

[Ueda et al., 2000] Ueda, N., Nakano, R., Ghahramani, Z., and Hinton, G. E. (2000). SMEM algorithm for mixture models. *Neural Computation*, 12(9):2109–2128. 23

[van den Oord et al., 2018] van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv:1807.03748*. 16

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc. 24

[Vinyals et al., 2014] Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. E. (2014). Grammar as a foreign language. In *Neural Information Processing Systems*. 38

[Viola and Jones, 2004] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154. 14

[Williams et al., 1995] Williams, C., Revow, M., and Hinton, G. (1995). Using a neural net to instantiate a deformable model. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 965–972. 13

[Williams and Agakov, 2002] Williams, C. K. I. and Agakov, F. V. (2002). Products of gaussians and probabilistic minor component analysis. *Neural Computation*, 14(5):1169–1182. 20

[Zemel et al., 1995] Zemel, R. S., Williams, C. K. I., and Mozer, M. C. (1995). Lending direction to neural networks. *Neural Networks*, 8(4):503–512. 3

GLOM suggests contrastive auxiliary loss to avoid feature collapse. Contrastive training methods need extra pairs of positive and negative samples, which is complicated and unstable. In RevCol, reversible transformations between columns provides lossless information propagation by nature. As for other multi-scale grid-like architectures such as *HRNets* (Wang et al., 2020), *DEQ models* (Bai et al., 2020) and *FPNs* (Lin et al., 2017; Tan et al., 2020), the design purpose of those models is to fuse multi-scale features rather than learn disentangled representations; therefore, in general they still follow the paradigm in Fig. 1 (a) – neither multiple entrances/exits nor reversible structures are employed. Based on those grid-like network topology, NAS based works (Ding et al., 2021; Wu et al., 2021; Liu et al., 2019; Ghiasi et al., 2019) search the optimized topology of network architectures for specific dataset. However, the RevCol architecture is not limit to specific tasks or datasets. With the reversible nature, our method maintains lossless information propagation and benefits for not only pre-training but also other down-stream tasks. Very recently, *RevBiFPN* (Chiley et al., 2022) comes up with an reversible variant of FPN, which is further employed in an HRNet-like architecture. Though our RevCol shares the similar idea of multi-scale reversible transformations with RevBiFPN, our work is done independently, which is derived from a different motivation of feature disentangling, and has much simpler architectures (*e.g.* free of reversible upsampling tower) and higher performances. We compare some of those models in Sec. 3.

2 METHOD

In this section, we introduce the design details of our *Reversible Column Networks (RevCol)*. Fig. 1 (b) illustrates the top-level architecture. Notice that for each column in RevCol, for simplicity we directly reuse existing structures such as *ConvNeXt* (Liu et al., 2022b), hence in the following subsections, we mainly focus on how to build the reversible connections between columns. In addition, we introduce an plug-and-play intermediate supervision on top of each column, which further improves the training convergence and feature quality.

2.1 MULTI-LEVEL REVERSIBLE UNIT

In our network, *reversible transformations* plays a key role in feature disentangling without information loss, whose insight comes from *Reversible Neural Networks* (Dinh et al., 2014; Chang et al., 2018; Gomez et al., 2017; Jacobsen et al., 2018; Mangalam et al., 2022). Among them, we first take a review of one representative work *RevNet* (Gomez et al., 2017). As shown in Fig. 2 (a), RevNet first partitions the input x into two groups, x_0 and x_1. Then for later blocks, for example, block t, it takes two anterior blocks' outputs x_{t-1} and x_{t-2} as input and generates the output x_t. The mapping of block t is *reversible*, i.e. x_{t-2} can be reconstructed by two posterior blocks x_{t-1} and x_t. Formally, the forward and *inverse* computation follow the equations [†]

$$\begin{aligned} Forward: x_t &= \boldsymbol{F}_t(x_{t-1}) + \gamma x_{t-2} \\ Inverse: x_{t-2} &= \gamma^{-1}[x_t - \boldsymbol{F}_t(x_{t-1})], \end{aligned} \tag{1}$$

where $\boldsymbol{F}_t$ denotes an arbitrary non-linear operation analogous to those residual functions in standard *ResNets*; γ is a simple reversible operation (*e.g.* channel-wise scaling), whose inverse is denoted by γ^{-1}. As mentioned in the introduction, the above formulation involves too strong constraint on the feature dimensions, i.e. $x_t, x_{t+2}, x_{t+4}, ...$ have to be equal sized, which is not flexible in architecture design. That is why RevNets (Gomez et al., 2017) introduce some non-reversible down-sampling blocks between reversible units, hence the whole network is not fully reversible. More importantly, we find there is no clear way to directly employ Eq. 1 to bridge the columns in Fig. 1 (b).

To address the issue, we generalize Eq. 1 into the following form:

$$\begin{aligned} Forward: x_t &= \boldsymbol{F}_t(x_{t-1}, x_{t-2}, ..., x_{t-m+1}) + \gamma x_{t-m} \\ Inverse: x_{t-m} &= \gamma^{-1}[x_t - \boldsymbol{F}_t(x_{t-1}, x_{t-2}, ..., x_{t-m+1})], \end{aligned} \tag{2}$$

where m is the *order* of the recursion ($m \geq 2$). Clearly, the extension is still reversible. Then we partition every m feature maps into a group: $(x_1, x_2, \ldots, x_m), (x_{m+1}, x_{m+2}, \ldots, x_{2m}), \ldots$. Given

[†] In Gomez et al. (2017), the proposed reversible equations are formulated as $y_1 = x_1 + \mathcal{F}(x_2)$ and $y_2 = x_2 + \mathcal{G}(y_1)$. While in this paper, we reformulate those notations $y_2, y_1, x_2, x_1, \mathcal{G}, \mathcal{F}$ as $x_t, x_{t-1}, x_{t-2}, x_{t-3}, \boldsymbol{F}_t, \boldsymbol{F}_{t-1}$, respectively, in order to better illustrate the relation between building block t and $t-1$. It is easy to prove the two formulations are equivalent.

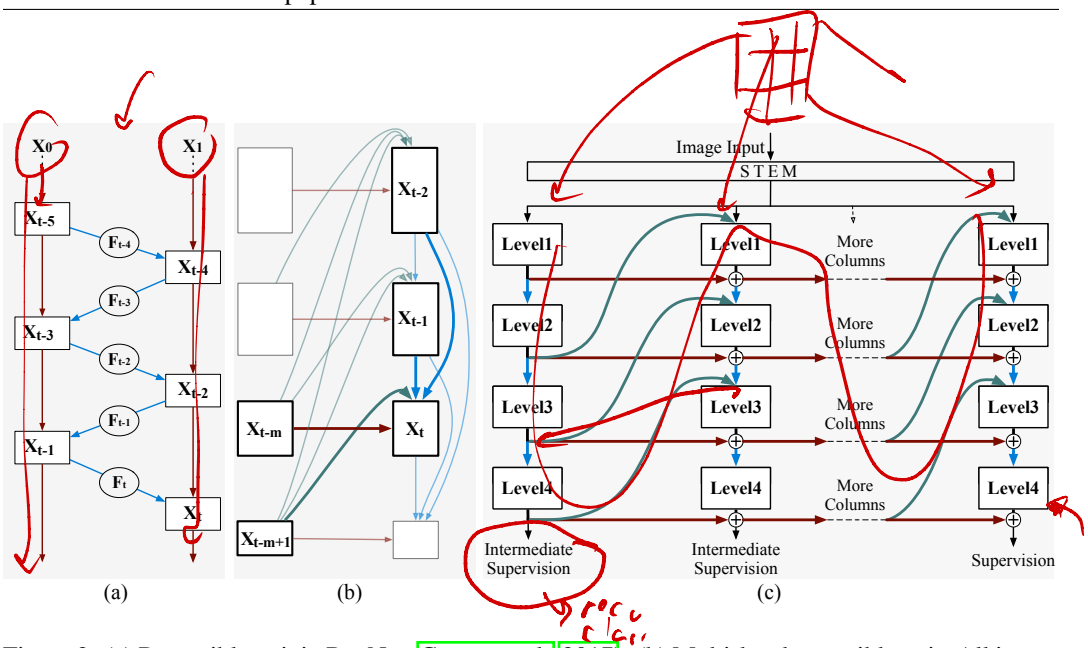


Figure 2: (a) Reversible unit in RevNet (Gomez et al., 2017). (b) Multi-level reversible unit. All inputs for level t are highlighted. (c) An overview of the whole reversible column network architecture, with simplified multi-level reversible unit.

the features within any of the group, we can easily compute the features in other groups recursively according to Eq. 2. Compared with the original form, Eq. 2 has the following two nice properties:

- The constraint on the feature map sizes is greatly relaxed if m is relatively large. Notice that Eq. 1 does not require feature maps *within* each group to be equal sized; such constraint only exist between groups. Therefore, we can use tensors of different shape to represent features of different semantic levels or different resolutions.
- Eq. 2 can easily cooperate with existing network architectures, even though the latter is not reversible. For example, we can assign m feature maps in a standard *ResNet* to represent the feature maps within a group $(x_t, x_{t+1}, \ldots, x_{t+m-1})$, which is still compatible with Eq. 2 since ResNet can be viewed as a part of $(\boldsymbol{F}_t, \boldsymbol{F}_{t+1}, \ldots, \boldsymbol{F}_{t+m-1})$ respectively. Thus the whole network is still reversible.

Therefore, we can reorganize Eq. 2 into a *multi-column* fashion, as shown in Fig. 2 (b). Each column is composed of m feature maps within a group, as well as their mother network. We name it *multi-level reversible unit*, which is the basic component of our RevCol as in Fig. 1 (b).

2.2 Reversible Column Architecture

2.2.1 Macro Design

As discussed in the introduction (see Fig. 1 (b)), our network *RevCol* is composed of multiple subnetworks with *reversible connections* to perform feature disentangling. Fig. 2 (c) elaborates the architecture design. Following the common practice of recent models (Dosovitskiy et al., 2020; Liu et al., 2022b), first the input image is split into non-overlapping patches by a patch embedding module. After that, patches are fed into each subnetwork (column). Columns can be implemented with any conventional single-column architectures, *e.g. ViT* (Dosovitskiy et al., 2020) or *ConvNeXt* (Liu et al., 2022b). We extract four-level feature maps from each column to propagate information between columns; for example, if the columns are implemented with widely-used hierarchical networks (Liu et al., 2021; He et al., 2016; Liu et al., 2022b), we can simply extract multi-resolution features from the output of each stage. For classification tasks, we only use feature map of the last level (Level 4) in the last column for rich semantic information. For other down-stream tasks like object detection and semantic segmentation, we use feature maps of all the four levels in the last column as they contain both low-level and semantic information.

To implement the reversible connections between columns, we adopt the *multi-level reversible unit* proposed in Eq. 2, but in a simplified fashion: rather than take $(m-1)$ inputs for each non-linear

operation $\boldsymbol{F}_t(\cdot)$, we use only one low-level feature x_{t-1} at the current column and one high-level feature x_{t-m+1} at the previous column as the input. The simplification does not break the reversible property. We find more inputs bring minor accuracy gain but consume much more GPU resources. Thus Eq. 2 is simplified as:

$$\begin{aligned} Forward: x_t &= \boldsymbol{F}_t(x_{t-1}, x_{t-m+1}) + \gamma x_{t-m} \\ Inverse: x_{t-m} &= \gamma^{-1}[x_t - \boldsymbol{F}_t(x_{t-1}, x_{t-m+1})]. \end{aligned} \tag{3}$$

Compared with conventional architectures, the *macro design* of our RevCol has the following three properties or advantages:

Feature disentangling. In RevCol, the lowest level of each column maintains low-level features as it is close to the input, while the highest level in the last column is highly semantic because it is directly connected to the supervision. Therefore, information in different levels is gradually *disentangled* during the (lossless) propagation between columns – some feature maps are more and more semantic and some maintain to be low-level. Detailed analyses are presented in Appendix E. The property brings many potential advantages, for instance, more flexible to downstream tasks which rely on both high-level and low-level features. We argue that *reversible connection* plays a key role in the disentangling mechanism – some previous works like *HRNet* (Wang et al., 2020) involve multi-level feature fusion but without reversible connection, which may suffer from information loss and lead to inferior performances in our experiments (see Section 3.5.2).

Memory saving. The training of conventional networks takes a lot of memory footprint to store the activations during forward propagation as the demand of gradient computation. While in our RevCol, since the connections between columns are *explicitly* reversible, during the back-propagation we can reconstruct the required activations *on the fly* from the last column to the first, which means we only need to maintain activations from *one* column in memory during training. In Section 3.5.4, we demonstrate RevCol costs roughly $\mathcal{O}(1)$ additional memory with the increase of column numbers.

New scaling factor for big models. In RevCol architecture, column number serves as a new dimension in addition to depth (number of blocks), and width (channels of each block) in vanilla single-column CNNs or ViTs. Increasing column numbers has similar income as increasing both width and depth in certain range.

2.2.2 Micro Design

We employ *ConvNeXt* blocks (Liu et al., 2022b) to implement each column in our network by default; other architectures, such as *transformers*, are also applicable (see Appendix B for details). We make a few modifications to make ConvNeXt compatible with our macro architecture:

Fusion module. As shown in Fig. 5, in each level of original ConvNeXt, the inputs are first down-sampled in a patch-merging block. Then the outputs are passed through a bunch of residual blocks. In RevCol, we introduce a fusion module to fuse the feature maps from the current and previous columns (refer to Fig. 2 (c), green and blue connections). We modify the original patch-merging block in ConvNeXt by putting the LayerNorm after the patch-merging convolution rather than before. Channel numbers are doubled in patch-merging convolution. We also introduce an up-sampling block, which is composed of a linear channel mapping layer, a LayerNorm normalization and a feature map interpolation layer. We halve the channel numbers in linear channel mapping layer. The outputs of the two blocks are summed up and then passed to the residual blocks followed by.

Kernel size. In RevCol we revise the 7×7 convolutions in original ConvNeXt (Liu et al., 2022b) to 3×3 by default, mainly to speed up the training. Increasing kernel size further obtains more accuracy, but not very much, partly because the our multi-column design enlarges the effective receptive field. Please refer to Section 3.5.5 for more details.

Reversible operation $\boldsymbol{\gamma}$. We adopt a learnable reversible channel-wise scaling as reversible operation γ to keep the network stable. Each time the features are summed up in forward of Eq. 3,

the magnitude grows larger, which makes the training process unstable. Using a learnable scaling can suppress the magnitude of features. During training, we truncate the absolute value of γ so that it will never be smaller than $1e^{-3}$, because the numerical error could become large in the reverse computation when γ is too small.

2.3 Intermediate Supervision

Though *multi-level reversible unit* is able to maintain information during column iteration, the down-sample block still can discard information *inside* column. Features at the end of the front columns is too close to the final output, for reversible connections simply do scaling and summation. Such information loss leads to inferior performance. Similar problem also happens when using deeply-supervised method (Lee et al., 2015; Szegedy et al., 2015).

To mitigate the problem of information collapse, we propose an intermediate supervision method which adds additional supervision into front columns. For features in front columns, we hope to keep the mutual information between features and the input image as much as possible, so that the network discard less information within columns. Consider *RevCol* gradually disentangle semantic and low-level information, extracting and leveraging the task-relevant information can further boost the performance. Therefore, we need to maximize the lower bound of mutual information between features and the prediction.

Inspired by Wang et al. (2021), we add two auxiliary heads to last level features (Level 4). One is a decoder (He et al., 2022) which reconstructs the input image, the other is a linear classifier. The linear classifier can be trained in a regular classification fashion with the *cross-entropy (CE)* loss. The parameters of decoder are optimized by minimizing the *binary cross-entropy (BCE)* reconstruction loss. Compared with commonly used *L1* and *L2* loss, interpreting the distribution of reconstructed logits and input image as *bit probabilities (Bernoullis)* outputs smoother value, which makes it more compatible with CE Loss.

For intermediate supervision at one column, the compound loss is the weighted summation of the above two loss. Note that supervision heads may not be added to all columns. For all the variants of RevCol, we set the number of compound loss to 4 empirically (eg. for a 8 column RevCol, the supervision heads are added to column 2, 4, and 6, and 8).

The total loss $\boldsymbol{L}$ in is the summation of all compound loss:

$$\boldsymbol{L} = \sum_{i=1}^{n} (\alpha_i \mathcal{L}_{\text{BCE}} + \beta_i \mathcal{L}_{\text{CE}}) \tag{4}$$

n denotes the total number of compound loss. $\mathcal{L}_{\text{BCE}}$ and $\mathcal{L}_{\text{CE}}$ denotes BCE loss and CE loss correspondingly. α_i and β_i are changed linearly with the compound loss number. When the compound loss is added in earlier columns, we use larger value α_i and smaller value β_i to keep $I(\boldsymbol{h}, x)$. In later columns, value α_i decreases and β_i increases, which helps boost the performance.

3 Experiments

We construct different *RevCol* variants, RevCol-T/S/B/L, to be of similar complexities to Swin transformers and ConvNeXts. We also build a larger RevCol-XL and RevCol-H to test the scaling up capability. These variants adopt different channel dimension C, blocks in each column B and column numbers *COL*. The configuration hyper-parameters of these model variants are:

- RevCol-T: $C = (64, 128, 256, 512)$, $B = (2, 2, 4, 2)$, $COL = 4$
- RevCol-S: $C = (64, 128, 256, 512)$, $B = (2, 2, 4, 2)$, $COL = 8$
- RevCol-B: $C = (72, 144, 288, 576)$, $B = (1, 1, 3, 2)$, $COL = 16$
- RevCol-L: $C = (128, 256, 512, 1024)$, $B = (1, 2, 6, 2)$, $COL = 8$
- RevCol-XL: $C = (224, 448, 896, 1792)$, $B = (1, 2, 6, 2)$, $COL = 8$
- RevCol-H: $C = (360, 720, 1440, 2880)$, $B = (1, 2, 6, 2)$, $COL = 8$

We conduct image classification on *ImageNet* dataset (Deng et al., 2009; Ridnik et al., 2021). We also test our models on the downstream object detection task and semantic segmentation task on

Table 1: **ImageNet classification results.** We compare our models with state-of-the-art ○ Vision Transformers and ● CNNs that have comparable FLOPs and parameters. ↑ denotes models fine-tuning using image size larger than 224^2. We report the top-1 accuracy on the validation set of ImageNet as well as the number of parameters and FLOPs. Our models are highlighted in gray.

Model	Image Size	Params (M)	FLOPs (G)	Top-1 Acc.
ImageNet-1K trained models				
○ Swin-T (Liu et al.)	224^2	28	4.5	81.3
○ DeiT-S (Touvron et al.)	224^2	22	4.6	79.8
○ Rev-ViT-S (Mangalam et al.)	224^2	22	4.6	79.9
● RevBiFPN-S3 (Chiley et al.)	288^2	20	3.3	81.1
● EfficientNet-B4 (Tan & Le)	380^2	19	4.2	82.9
● ConvNeXt-T (Liu et al.)	224^2	29	4.5	82.1
● RevCol-T	224^2	30	4.5	82.2
○ Swin-S (Liu et al.)	224^2	50	8.7	83.0
○ MViTv1-B (Fan et al.)	224^2	37	7.8	83.0
○ T2T-ViT-19 (Yuan et al.)	224^2	39	8.4	81.4
● RevBiFPN-S4 (Chiley et al.)	320^2	49	10.6	83.0
● EfficientNet-B5 (Tan & Le)	456^2	30	9.9	83.6
● ConvNeXt-S (Liu et al.)	224^2	50	8.7	83.1
● RevCol-S	224^2	60	9.0	83.5
○ Swin-B (Liu et al.)	224^2	89	15.4	83.5
○ DeiT-B (Touvron et al.)	224^2	86	17.5	81.8
○ Rev-ViT-B (Mangalam et al.)	224^2	87	17.6	81.8
● RepLKNet-31B (Ding et al.)	224^2	79	15.3	83.5
● RevBiFPN-S5 (Chiley et al.)	352^2	82	21.8	83.7
● EfficientNet-B6 (Tan & Le)	528^2	43	19.0	84.0
● ConvNeXt-B (Liu et al.)	224^2	88	15.4	83.8
● RevCol-B	224^2	138	16.6	84.1

Model	Image Size	Params (M)	FLOPs (G)	Top-1 Acc.
ImageNet-22K pre-trained models (ImageNet-1K fine-tuned)				
○ Swin-B (Liu et al.)	224^2	88	15.4	85.2
○ Swin-B↑ (Liu et al.)	384^2	88	47.0	86.4
○ ViT-B↑ (Dosovitskiy et al.)	384^2	86	55.4	84.0
● RepLKNet-31B (Ding et al.)	224^2	79	15.3	85.2
● RepLKNet-31B↑ (Ding et al.)	384^2	79	45.1	86.0
● ConvNeXt-B (Liu et al.)	224^2	89	15.4	85.8
● ConvNeXt-B↑ (Liu et al.)	384^2	89	45.1	86.8
● RevCol-B	224^2	138	16.6	85.6
● RevCol-B↑	384^2	138	48.9	86.7
○ Swin-L (Liu et al.)	224^2	197	34.5	86.3
○ Swin-L↑ (Liu et al.)	384^2	197	103.9	87.3
○ ViT-L↑ (Dosovitskiy et al.)	384^2	307	190.7	85.2
● RepLKNet-31L (Ding et al.)	384^2	172	96.0	86.6
● ConvNeXt-L (Liu et al.)	224^2	198	34.4	86.6
● ConvNeXt-L↑ (Liu et al.)	384^2	198	101.0	87.5
● RevCol-L	224^2	273	39.0	86.6
● RevCol-L↑	384^2	273	116.0	87.6
● ConvNeXt-XL↑ (Liu et al.)	384^2	350	179.0	87.8
● RevCol-XL↑	384^2	834	350.0	88.2
Extra data pre-trained models (ImageNet-1K fine-tuned)				
● RevCol-XL↑	384^2	834	350.0	89.4
● RevCol-H↑	640^2	2158	2537	90.0

commonly used *MS-COCO* (Lin et al., 2014) and *ADE20k* (Zhou et al., 2017b) dataset. Training and fine-tuning settings please refer to Appendix D. Furthermore, we show the performance of RevCol with transformer on vision and language tasks (shown in Appendix B).

3.1 Image Classification

On ImageNet (1.28M images) (Deng et al., 2009) dataset, we train RevCol for 300 epochs with intermediate supervision. Hyperparameters, augmentation and regularization strategies follows Liu et al. (2022b) We also pre-train our models on the larger ImageNet-22K dataset (Ridnik et al., 2021), which contains 14.2 million images.

In Tab. 1, we compare our RevCol variants with commonly used recent *Transformers* and *CNNs* on ImageNet-1k validation set. Our models outperforms a large number of vanilla single-column CNNs and Transformers with similar complexities. For example, RevCol-S achieve 83.5% Top-1 accuracy, outperform ConvNeXt-S by 0.4 points. When pre-trained with larger ImageNet-22K dataset, RevCol-XL achieves 88.2% Top-1 accuracy. As RevCol maintains some task-irrelevant low-level information in classification pre-training, relaxing the constraint of params and FLOPs and enlarging dataset size can further boost our models' performance. To further test the scaling up effectiveness of large dataset, we build a 168-million-image semi-labeled dataset (see Appendix C). With extra data pre-training and ImageNet-1k fine-tuning, our RevCol-H achieves **90.0% top-1 accuracy**. Our results further demonstrate with RevCol, CNN models can also share the dividends of large model and massive data pre-training.

3.2 Object Detection

We evaluate our proposed RevCol on object detection task. Experiments are conducted on the MS-COCO dataset using the Cascade Mask R-CNN (Cai & Vasconcelos, 2019) framework. We also finetune our largest model RevCol-H with HTC++ (Chen et al., 2019) and DINO (Zhang et al., 2022a) Framework.

Table 2: **Object detection results on MS-COCO dataset with different backbones.** We report box AP and mask AP with single scale testing on COCO minival set. FLOPs are measured under input sizes of (1280, 800).

Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	Params	FLOPs
ImageNet-1K pre-trained								
○ Swin-T (Liu et al.)	50.5	69.3	54.9	43.7	66.6	47.1	86M	745G
● ConvNeXt-T (Liu et al.)	50.4	69.1	54.8	43.7	66.5	47.3	86M	741G
● RevCol-T	50.6	68.9	54.9	43.8	66.7	47.4	88M	741G
○ Swin-S (Liu et al.)	51.8	70.4	56.3	44.7	67.9	48.5	107M	838G
● ConvNeXt-S (Liu et al.)	51.9	70.8	56.5	45.0	68.4	49.1	108M	827G
● RevCol-S	52.6	71.1	56.8	45.5	68.8	49.0	118M	833G
○ Swin-B (Liu et al.)	51.9	70.9	56.5	45.0	68.4	48.7	145M	982G
● ConvNeXt-B (Liu et al.)	52.7	71.3	57.2	45.6	68.9	49.5	146M	964G
● RepLKNet-B (Ding et al.)	52.2	-	-	45.2	-	-	137M	965G
● RevCol-B	53.0	71.4	57.3	45.9	69.1	50.1	196M	988G
ImageNet-22K pre-trained								
○ Swin-B (Liu et al.)	53.0	71.8	57.5	45.8	69.4	49.7	145M	982G
● ConvNeXt-B (Liu et al.)	54.0	73.1	58.8	46.9	70.6	51.3	146M	964G
● RepLKNet-B (Ding et al.)	53.0	-	-	46.3	-	-	137M	965G
● RevCol-B	55.0	73.5	59.7	47.5	71.1	51.8	196M	988G
○ Swin-L (Liu et al.)	53.9	72.4	58.8	46.7	70.1	50.8	253M	1382G
● ConvNeXt-L (Liu et al.)	54.8	73.8	59.8	47.6	71.3	51.7	255M	1354G
● RepLKNet-L (Ding et al.)	53.9	-	-	46.5	-	-	229M	1321G
● RevCol-L	55.9	74.1	60.7	48.4	71.8	52.8	330M	1453G
Extra data pre-trained								
● RevCol-H (HTC++)	61.1	78.8	67.0	53.0	76.3	58.7	2.41G	4417G
● RevCol-H (Objects365+DINO)	63.8	81.8	70.2	-	-	-	2.18G	4012G

In Tab. 2, we compare the AP_{box} and AP_{mask} with Swin/ConvNeXt in variant sizes on COCO validation set. We find RevCol models surpass other counterparts with similar computation complexities. Information retained in pre-training helps RevCol models acheieve better results in down-stream tasks. When the model size grows larger, this advantage becomes more remarkable. After finetuning under Objects365(Shao et al., 2019) dataset and DINO (Zhang et al., 2022a) framework, our largest model RevCol-H achieves **63.8%** AP_{box} on COCO detection minival set.

Table 3: **Semantic segmentation result on ADE20k dataset with different backbones.** we report mIoU results with single/multi-scale testing. FLOPs are measured under input sizes of (2048, 512), (2560, 640) for IN-1K and IN-22K pre-trained models respectively.

Backbone	crop size	$mIoU_{ss}$	$mIoU_{ms}$	Params	FLOPs
ImageNet-1K pre-trained					
○ Swin-T (Liu et al.)	512^2	44.5	45.8	60M	945G
● ConvNeXt-T (Liu et al.)	512^2	46.0	46.7	60M	939G
● RevCol-T	512^2	47.4	47.6	60M	937G
○ Swin-S (Liu et al.)	512^2	47.6	49.5	81M	1038G
● ConvNeXt-S (Liu et al.)	512^2	48.7	49.6	82M	1027G
● RevCol-S	512^2	47.9	49.0	90M	1031G
○ Swin-B (Liu et al.)	512^2	48.1	49.7	121M	1188G
● RepLKNet-B (Ding et al.)	512^2	49.9	50.6	112M	1170G
● ConvNeXt-B (Liu et al.)	512^2	49.1	49.9	122M	1170G
● RevCol-B	512^2	49.0	50.1	122M	1169G
ImageNet-22K pre-trained					
○ Swin-B (Liu et al.)	640^2	50.3	51.7	121M	1841G
● RepLKNet-B (Ding et al.)	640^2	51.5	52.3	112M	1829G
● ConvNeXt-B (Liu et al.)	640^2	52.6	53.1	122M	1828G
● RevCol-B	640^2	52.7	53.3	122M	1827G
○ Swin-L (Liu et al.)	640^2	52.1	53.5	234M	2468G
● RepLKNet-L (Ding et al.)	640^2	52.4	52.7	207M	2404G
● ConvNeXt-L (Liu et al.)	640^2	53.2	53.7	235M	2458G
● RevCol-L	640^2	53.4	53.7	306M	2610G
Extra data pre-trained					
● RevCol-H	640^2	57.8	58.0	2421M	-
● RevCol-H + Mask2Former	640^2	60.4	61.0	2439M	-

Table 4: **System-level comparison of state-of-the-art visual foundation models with large-scale pretraining.** We include ○ Vision Transformers, ● CNNs, and ● hybrid architectures pretrained either *unsupervised* or *supervised* on *image-only* and *vision-language* datasets. COCO scores marked with † means intermediate fine-tuned on extra data like Object365 (Shao et al., 2019).

Model	Params	Dataset		ImageNet	COCO test-dev			ADE20K		
		Images	Annotation	1k	Detector	AP_{box}	AP_{mask}	Segmenter	mIoU	+ms
○ SwinV2-G	3.0 G	70 M	labeled	90.2	HTC++	63.1†	54.4†	UperNet	59.3	59.9
○ BEiT3	1.0 G	35 M	labeled & image-text	89.6	ViTDet	63.7†	54.8†	Mask2Former	62.0	62.8
● Florence	0.9 G	900 M	image-text	90.1	DyHead	62.4	-	-	-	-
● RevCol-H	2.1 G	168 M	semi-labeled	90.0	DINO	63.6†	-	Mask2Former	60.4	61.0

3.3 Semantic Segmentation

We also evaluate RevCol backbones on the ADE20K semantic segmentation task with *UperNet* (Xiao et al., 2018) framework. We do not use intermediate-supervision in down-stream fine-tune process. To further explore our model's capacity and reach the leading performance, we utilize recent segmentation framework *Mask2Former* (Cheng et al., 2022), and adopt the same training settings.

In Tab. 3, we report validation *mIoU* with single-scale and multi-scale flip testing. RevCol models can achieve competitive performance across different model capacities, further validating the effectiveness of our architecture design. It's worth noting that when use Mask2Former detector and extra pretraining data, RevCol-H achieves an mIoU of 61.0%, which shows feasible scalability towards large-scale vision applications.

3.4 System-level Comparison with SOTA Foundation Models

Foundation models (Kolesnikov et al., 2020; Radford et al., 2021; Yuan et al., 2021b) are general-purpose backbones pre-trained on massive and divergent data source. They can adapt to various down-stream tasks with limited domain-specific data. We show comparison among various public *state-of-the-art (SOTA)* foundation models including Vision Transformers and Vision-Language models, namely, *SwinV2* (Liu et al., 2022a), *BEiT3* (Wang et al., 2022), and *Florence* (Yuan et al., 2021b). As shown in Tab. 4, though our RevCol-H is *purely convolutional and pre-trained on single modality dataset*, the results on different tasks demonstrate remarkable generalization ability of RevCol with large scale parameters.

3.5 More Analysis Experiments

3.5.1 Performance Gain of Reversible Columns Architecture

In this section, we evaluate the performance gain of using reversible columns. In the first experiment, we fix a single column's structure and FLOPs then simply add more columns to scale large and test the performance. At the same time, we plot the vanilla single-column models with similar model sizes. As depicted in Fig. 3, compared to single-column models, using multi-column reversible architecture always gets better performance under same FLOPs constraint. Besides, within a certain range, scaling up RevCol in terms of increasing column numbers can have similar gains compared to scaling up with both block numbers(depth) and channel numbers(width) in single-column models. In the second experiment, we limit the model size to about 4.5G FLOPs and test model variants with different column numbers. In other words, we gradually add more columns and scale down the single column size at the same time. Results are shown in Tab. 5, we notice that adopt column number at the range of 4 to 12 can maintain the model's performance, then further more column models suffer from performance degradation. We believe the reason is the width and depth in a single column are too low to keep representation ability.

3.5.2 Reversible Networks vs. Non-Reversible Networks

In this section, we ablate different design patterns of reversible connections. First, we build a non-reversible multi-column network using the fusion module of HRNet. Second, we build another single column reversible ConvNeXt using the design of RevNet as shown in Fig. 2(a). We compare the two

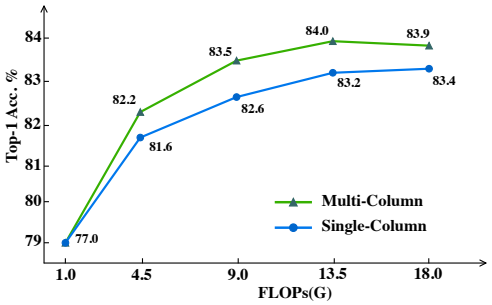


Figure 3: ImageNet-1K performance of maintaining a constant FLOPs of a single column and adding more columns.

Table 5: ImageNet 1K performances of various number of columns in RevCols under the similar computational budget.

# column	Params	FLOPs	FLOPs per col.	Top-1 Acc.
1	28M	4.4G	4.40G	81.9
4	30M	4.5G	1.12G	82.2
8	34M	4.7G	0.59G	82.3
12	33M	4.4G	0.35G	82.2
20	35M	4.2G	0.21G	81.0

designs with our RevCols. The evaluation result is shown in Tab. 6. The non-reversible multi-column network suffers from information loss during propagation, which could result in lower accuracy. The reversible single-column network maintains information during propagation, but lack the superiority of multi-level fusion. This experiment further indicates the effectiveness of combining the reversible design with multi-column networks.

Table 6: Performance comparison on ImageNet-1K of different design patterns. Row-1 represents HRNet style network w/o *reversible connections*. Row-2 represents RevNet style network w/o *multi-column fusions*. Row-3 are our proposed RevCols.

rev. conn.	*multi-col.*	Params	FLOPs	Acc.
	✓	35M	4.9G	78.8
✓		34M	4.5G	81.6
✓	✓	30M	4.5G	82.2

Table 7: Performance comparison between models with and without *intermediate supervision*. Results are reported on ImageNet-1K and COCO dataset. We use 1× training schedule on COCO detection task.

Model	*inter. sup.*	Top-1 Acc.	AP_{box}	AP_{mask}
RevCol-T	✗	81.4	48.3	41.8
RevCol-T	✓	82.2 (+0.8)	48.8 (+0.6)	42.2 (+0.4)
RevCol-S	✗	83.0	50.7	43.8
RevCol-S	✓	83.5 (+0.5)	51.1 (+0.4)	43.8 (+0.0)
RevCol-B	✗	83.2	51.2	44.2
RevCol-B	✓	84.1 (+0.9)	51.6 (+0.4)	44.2 (+0.0)

3.5.3 Performance Gain of Using Intermediate Supervision

In this section, we evaluate the performance of RevCol-T/S/B with and without intermediate supervision on ImageNet-1K. We also evaluate the object detection task performance using 1× training schedule on MS-COCO dataset. Other settings remain the same. From the validation results in Tab. 7, models trained with intermediate supervision achieves 0.5% to 0.9% better top-1 accuracy. Besides, intermediate supervision also benefits down-stream tasks, which further demonstrates its effectiveness.

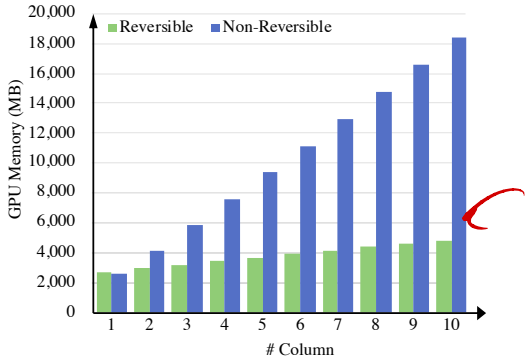


Figure 4: GPU Memory Consumption *vs*. Model size

Table 8: Performance of models with larger kernel convolution.

Kernel Size	FLOPs	Top-1 Acc	AP_{box} 1×	AP_{mask} 1×
3	4.5G	82.2	48.8	42.2
5	4.5G	82.5	49.5	42.6
7	4.6G	82.5	49.3	42.4
11	4.6G	82.5	49.9	42.7

3.5.4 GPU MEMORY CONSUMPTION VS MODEL SIZE

Fig. 4 plots the GPU memory consumption with the scaling of model size. We fix the computation complexity of a single column to 1G FLOPs and increase column number. Meanwhile, we measure the memory consumption in training process which includes the forward and backward propagation. Our experiments are conducted on Nvidia Tesla V100 GPU under batch-size 64, FP16 precision and PyTorch implementation. With the increment of column number, we can see RevCol keeps an $\mathcal{O}(1)$ GPU memory consumption, while non-reversible architecture's memory consumption increase linearly with column number. Note that our RevCol does not keep strictly the same GPU memory consumption as column number increase, as reversible networks need to back-up the operation weights in need for calculating gradients and the re-construction of feature maps in backward propagation.

3.5.5 ABLATION OF KERNEL SIZE IN CONVOLUTIONS

In original ConvNeXt, large kernel convolution achieves in better performance. We conduct experiments in RevCol-T. As shown in Tab. 8, for 4 column models, using 5×5 convolution increase the ImageNet-1k Top-1 accuracy by 0.3% and the COCO AP_{box} by 0.7 for RevCol-T model. Further increasing kernel size obtains more accuracy in down-stream tasks, but not too much. We consider the RevCol design already enlarges the effective receptive field and this limit the accuracy gain of using large kernel convolution. On the other hand, 3×3 convolution enjoys the merits of efficiency and stability in (pre)training. Therefore, we adopt kernel 3 in all RevCol models.

4 RELATED WORKS

4.1 DISENTANGLE REPRESENTATION LEARNING AND PART-WHOLE HIERARCHY

A disentangled representation is generally described as one which separates the factors of variation, explicitly representing the important attributes of the data (Desjardins et al., 2012; Bengio et al., 2013). Desjardins et al. (2012); Kulkarni et al. (2015); Higgins et al. (2017); Chen et al. (2016); Karras et al. (2019) seek to learn disentangled representations through generative models. Locatello et al. (2019) points out that unsupervised learning of disentangled representations is fundamentally impossible without inductive biases both on the considered learning approaches and the datasets. The recent proposal of *GLOM* (Hinton, 2021) gives an idea of representing a part-whole hierarchy by a weight-sharing columns. The GLOM architecture provides an interpretable part-whole hierarchies for deep neural network (Garau et al., 2022). In RevCol, we adopt the design of using columns, but not modeling the process of formulating islands. On the contrary, our column iteration process maintains both low-level and high-level information and gradually disentangle them. Rather than using self-supervised methods, RevCol can be trained with supervision end-to-end.

4.2 REVERSIBLE NETWORKS

Gomez et al. (2017) firstly propose *RevNet* that allow back propagation without saving intermediate activations. The reversible design remarkably saves the training cost, since it keep $\mathcal{O}(1)$ GPU memory consumption as model depth scaling up. Jacobsen et al. (2018) propose a fully reversible network that can reverse back to the input without any information loss. Chang et al. (2018) develop a theoretical framework on stability and reversibility of deep neural network and derive reversible networks that can go arbitrarily deep. Mangalam et al. (2022) expand the reversible network scope from CNNs to Transformers. *RevBiFPN* (Chiley et al., 2022), a concurrent work of ours, add the reversible connections to *BiFPN* (Tan et al., 2020) network. Our RevCol maintains the information without loss inside each column rather than the whole BiFPN network in RevBiFPN.

5 CONCLUSION

In this paper, we propose RevCol, a reversible column based foundation model design paradigm. During the lossless propagation through columns, features in RevCol are learned to be gradually disentangled and the total information is still maintained rather than compressed. Our experiments suggests that RevCol can achieve competitive performance in multiple computer vision tasks. We hope RevCol could contribute to better performance in various tasks in both vision and language domains.

REFERENCES

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems*, 33:5238–5250, 2020.

Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: high quality object detection and instance segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 43(5):1483–1498, 2019.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4974–4983, 2019.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1290–1299, 2022.

Vitaliy Chiley, Vithursan Thangarasa, Abhay Gupta, Anshul Samar, Joel Hestness, and Dennis DeCoste. Revbifpn: The fully reversible bidirectional feature pyramid network. *arXiv preprint arXiv:2206.14098*, 2022.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 764–773, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entangling. *arXiv preprint arXiv:1210.5474*, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Mingyu Ding, Xiaochen Lian, Linjie Yang, Peng Wang, Xiaojie Jin, Zhiwu Lu, and Ping Luo. Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2982–2992, 2021.

Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11963–11975, 2022a.

Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11963–11975, 2022b.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6824–6835, 2021.

Nicola Garau, Niccolò Bisagno, Zeno Sambugaro, and Nicola Conci. Interpretable part-whole hierarchies and conceptual-semantic relationships in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13689–13698, 2022.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7036–7045, 2019.

Golnaz Ghiasi, Barret Zoph, Ekin D Cubuk, Quoc V Le, and Tsung-Yi Lin. Multi-task self-training for learning general representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 8856–8865, 2021.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. On the connection between local attention and dynamic depth-wise convolution. In *International Conference on Learning Representations*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

Geoffrey Hinton. How to represent part-whole hierarchies in a neural network. *arXiv preprint arXiv:2102.12627*, 2021.

Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

Peng-Tao Jiang, Chang-Bin Zhang, Qibin Hou, Ming-Ming Cheng, and Yunchao Wei. Layercam: Exploring hierarchical class activation maps for localization. *IEEE Transactions on Image Processing*, 30:5875–5888, 2021.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pp. 491–507. Springer, 2020.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pp. 3519–3529. PMLR, 2019.

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. *Advances in neural information processing systems*, 28, 2015.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pp. 562–570. PMLR, 2015.

Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 82–92, 2019.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12009–12019, 2022a.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022b.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pp. 4114–4124. PMLR, 2019.

Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. Weightnet: Revisiting the design space of weight networks. In *European Conference on Computer Vision*, pp. 776–792. Springer, 2020.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 181–196, 2018.

Karttikeya Mangalam, Haoqi Fan, Yanghao Li, Chao-Yuan Wu, Bo Xiong, Christoph Feichtenhofer, and Jitendra Malik. Reversible vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10830–10840, 2022.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pp. 1–9, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6301.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763. PMLR, 2021.

Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv:2104.10972*, 2021.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725. Association for Computational Linguistics (ACL), 2016.

Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8430–8439, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pp. 1–5. IEEE, 2015.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364, 2020.

Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022.

Yulin Wang, Zanlin Ni, Shiji Song, Le Yang, and Gao Huang. Revisiting locally supervised learning: an alternative to end-to-end training. *arXiv preprint arXiv:2101.10832*, 2021.

Bichen Wu, Chaojian Li, Hang Zhang, Xiaoliang Dai, Peizhao Zhang, Matthew Yu, Jialiang Wang, Yingyan Lin, and Peter Vajda. Fbnetv5: Neural architecture search for multiple tasks in one run. *arXiv preprint arXiv:2111.10007*, 2021.

Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 418–434, 2018.

Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9653–9663, 2022.

I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 558–567, 2021a.

Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, et al. Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*, 2021b.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3712–3722, 2018.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022a.

Yuanhan Zhang, Qinghong Sun, Yichun Zhou, Zexin He, Zhenfei Yin, Kun Wang, Lu Sheng, Yu Qiao, Jing Shao, and Ziwei Liu. Bamboo: Building mega-scale vision dataset continually with human-machine synergy, 2022b.

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017a.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 633–641, 2017b.

A MICRO DESIGN DETAILS

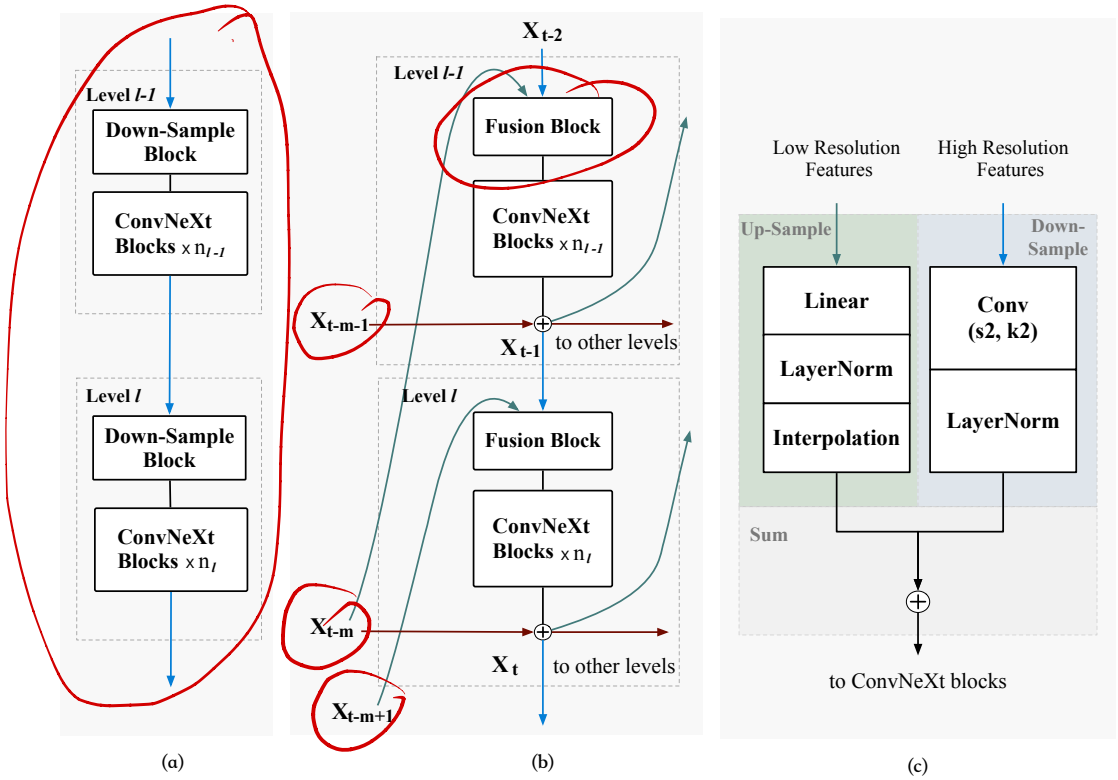


Figure 5: (a) Levels in ConvNeXt. Level l contains a patch merging down-sample block and n_l residual blocks. (b) Levels in RevCol. Level l is composed of a fusion module, n_l residual blocks and a reversible operation. Note that Level l takes features maps x_{t-1}, x_{t-m+1} and x_{t-m} as input. Feature maps x_{t-1} and x_{t-m+1} are fed into the fusion module and feature maps x_{t-m} are fed into the reversible operation. (c) Design of the fusion module.

In this section, we provide the architecture design details for RevCol. As depicted in Fig. 2 and Section 2.2, our RevCol contains multiple columns with reversible connections. Fig. 5 (a) shows the architecture of ConvNeXt. Note that we replace the 7×7 depth-wise convolution in ConvNeXt with 3×3, as described in Sec. 2.2.2. In Fig. 5 (b), we show in detail how to extend to our RevCol on the basis of ConvNeXt. First, we replace the down-sample block with a fusion block to fuse low-level representations in current column and high-level ones from the previous column, and Fig. 5 (c) shows the details of fusion block which contains up-sample and down-sample operations to handle different resolutions. Second, for each level, same-level representations from the previous column are added to current level's output and are ready to propagate as a whole. Thanks to the two modifications, feature maps from different hierarchies aggregate together to form the intermediate representation. In Fig. 5 (c), we use a `Linear-LayerNorm` followed by a nearest interpolation to up-sample low resolution features. A 2×2 kernel `Conv2d` with stride 2 down-samples the high resolution features, followed by a `LayerNorm` to balance the contributions of the two inputs.

B GENERALIZATION TO TRANSFORMERS

B.1 VISION TRANSFORMER MODELS

RevCol contains multiple light-weight sub-networks with reversible connections. In this paper, we adopt the ConvNext micro design by default except for multi-columns fusion and smaller convolution kernel as described in Sec. 2.2.2. However, the micro design of our RevCol is not limited to convolutional networks, but is also compatible with isotropic designing, such as the vanilla vision

transformer (ViT) (Dosovitskiy et al., 2020). In this section, we show the micro design of RevCol can generalized to vanilla ViT, namely RevCol-ViT, with promising experimental results.

net-ViT maintains the feature resolution in the reversible columns. Thus the patch merging blocks and up-sample blocks in the fusion modules are replaced with a simple linear projection with a post LayerNorm. We use the vanilla ViT building block instead of the ConvNext building block variant. The post LayerNorms and normalized dot-product attention are used in ViT blocks to stabilize training convergence, similar to Liu et al. (2022a). With the properties of isotropy, we evenly arrange the building blocks in each column. The configuration details of RevCol-ViT are:

- RevCol-ViT-S: $C = (224, 224, 224, 224)$, $B = (2, 2, 2, 2)$, $HEAD = 4$, $COL = 4$
- RevCol-ViT-B: $C = (384, 384, 384, 384)$, $B = (3, 3, 3, 3)$, $HEAD = 6$, $COL = 4$

Table 9: ImageNet-1K classification results. We compare our RevCol-ViT with state-of-the-art isotropic ○ Vision Transformers and ● CNNs that have comparable FLOPs and parameters.

Model	Image Size	Params	FLOPs	Top-1 Acc.
○ DeiT-S (Touvron et al., 2020)	224^2	22M	4.6G	79.8
● ConvNext-S (*iso.*) (Liu et al., 2022b)	224^2	22M	4.3G	79.7
○ **RevCol-ViT-S**	224^2	16M	4.6G	**80.6**
○ ViT-B (Dosovitskiy et al., 2020)	384^2	86M	55.4G	77.9
○ DeiT-B (Touvron et al., 2020)	224^2	86M	17.6G	81.7
○ Rev-ViT-B (Mangalam et al., 2022)	224^2	87M	17.6G	81.8
○ Rev-MViT-B (Mangalam et al., 2022)	224^2	39M	8.7G	82.5
● ConvNext-B (*iso.*) (Liu et al., 2022b)	224^2	87M	16.9G	82.0
○ **RevCol-ViT-B**	224^2	67M	18.8G	**82.7**

We use the same training setting with the anisotropic RevCol as described in Sec. 3.1, except that the intermediate supervision is discarded for simplicity and the stochastic depth rate is set as 0.2 for RevCol-B. We scale down the value of last linear projection layers in each FFN accroding to the network depth in initialization, same as BEiT (Bao et al., 2021). In Tab. 9, we compare the RevCol-ViT with vanilla ViT and other concurrent isotropic designs. Our RevCol-ViT surpasses vanilla vision transformer (77.9% for ViT and 81.7% for DeiT) and convolutional network ConvNeXt (82.0%) that have comparable model parameters and computational overhead on ImageNet-1k classification w.r.t. the top-1 accuracy.

B.2 Language Models

Considering the great success of applying transformer to computer vision, i.e., ViT (Dosovitskiy et al., 2020), we also made some exploration to generalize RevCol to natural language processing (NLP). Based on the design in Appendix B.1, we can easily apply the isotropic RevCol to language models with minor modification. To be specific, we replace the stem module in our RevCol with word embedding and positional encoding in transformer. Then, the RevCol can be plugged into the original transformer as an encoder. The output of the last column in RevCol will be used as the memory keys and values for the attention layers in decoder, just exactly the same as the original transformer.

We select the translation task to evaluate the potential of the RevCol in NLP. We run experiments on the WMT'16 English-German (En-De) dataset with 4.5M sentences and larger WMT'14 English-French dataset with 36M sentences. Each sentence is encoded by joint source and target byte pair encoding following Sennrich et al. (2016). The details of model architecture and the BLEU score are shown in Tab. 10.

All the dataset preparation and the training configurations follows Ott et al. (2018) and the open source project `fairseq`. The models were trained for 300K steps with batch-size of 28,672 tokens on En-De and 200K steps with batch-size of 86,016 on En-Fr. We discard the intermediate supervision for simplicity. As shown in Tab. 10, our RevCol outperforms vanilla transformer with comparable parameters on En-De (28.67 *vs*. 28.43) and En-Fr (43.40 *vs*. 43.07), which demonstrates the RevCol's applicability to NLP.

Table 10: BLEU score on newstest2014 for WMT English-German (En-De) and English-French (En-Fr) translation task. † indicates we re-run the experiments with `fairseq`.

Model	Encoder				Decoder				Params	Task	BLEU
	arch	d_{model}	d_{ff}	head	arch	d_{model}	d_{ff}	head			
Transformer$^{\dagger}_{big}$ (Vaswani et al., 2017)	$N = 6$	1024	4096	16	$N = 6$	1024	4096	16	209M 221M	En-De En-Fr	28.43 43.07
RevCol-Transformer	$B = (1,1,1,1)$ $COL = 4$	768	3072	12	$N = 6$	768	3072	12	200M 209M	En-De En-Fr	**28.67** **43.40**

B.3 Robustness of the number of columns

In the ablation analysis of the paper, we show that when fix the total FLOPs and add more columns of RevCol, the performance first increases and then get saturated. When the number of columns is extreme large, such as 20, the performance drop because of the representation ability of single column is limited. When the number of columns is usual, such as 4$\tilde{1}$2, the performances are similar, which verifies the setting robustness of the number of columns.

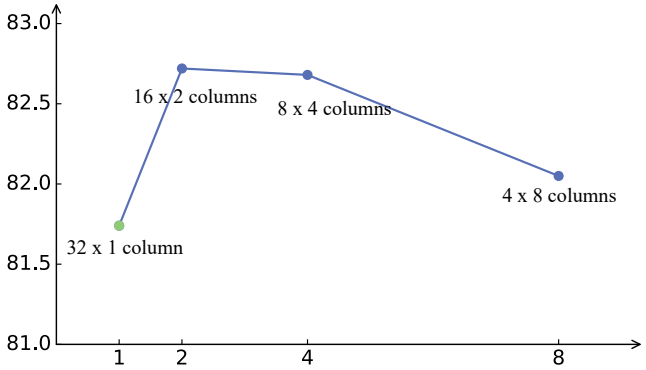


Figure 6: ImageNet top-1 accuracy of different variants of RevCol-ViT-B. Each variant has the same total number of residual blocks and channel dimension.

To further analyze the robustness of the number of columns, in this section, we build some RevCol-ViT-B variants (see Appendix B for more details). Each variant has the same number of residual blocks with the same channel dimension, but different number of columns. In other worlds, these variants have the same channel dimension and different depth of each columns and different number of columns. We use 32 residual blocks totally and maintain the FLOPs about 18G. Fig. 6 show the performance on ImageNet-1K of different variants. The number of columns are 1, 2, 4, and 8, accordingly the depth of each column are 32, 16, 8, and 4. The performance of single column variant is lower (similar to DeiT-B (Touvron et al., 2020)) because of the single column ViT can not maintain the information as multi reversible columns. The performance is decreasing when the number of columns became larger, because of the depth of each columns is not enough. This phenomenon indicates us that given a target FLOPs, the setting of the number of columns is robust unless the depth of each columns or channel dimension is too small.

C Semi-labeled Privately Collected Dataset for Large Models

C.1 Data Collection and Pseudo Label System

The dataset consists of around 168 million(M) images, 50M of which labeled and the remaining 118M unlabeled. The majority of labeled images come from public datasets, *e.g.* ImageNet, Places365 (Zhou et al., 2017a), and Bamboo (Zhang et al., 2022b). The others are web-crawled images annotated by in-door employees. Unlabeled images come from weakly-annotated image-text datasets like YFCC-100M (Thomee et al., 2016). We do not use text annotations.

In order to utilize images of different label domains and the massive unlabeled images, we employ a multi-target label system similar to Ding et al. (2022a) and Ghiasi et al. (2021). We adopt a semi-supervised learning strategy with ViTs, thus generating *pseudo* labels with continuously increased quality. We only store soft predictions with confidence higher than 1% to save storage. The final version of *pseudo* label we use are generated by a multi-head ViT-Huge teacher, which has an 89.0% ImageNet-1k accuracy.

C.2 Image Deduplication

Since the dataset contains large amount of unverified web-crawled images, there are probably validation or test images sneaking into our training dataset. Works like Mahajan et al. (2018) and Yalniz et al. (2019) all regard image deduplication an important procedure for fair experiments.

We first iterate over the entire dataset to filter out suspicious duplicates together with the corresponding test images based on their *pseudo* label distance. This brings more than 10,000 images with high suspicion. We look at these image pairs and finally find about 1,200 exact-duplicates and near-duplicates. Fig. 7 shows some examples of the near-duplicates, which are difficult to detect. Never the less, training a model without removing these duplicates gives less than 0.1% accuracy gain on ImageNet-1k in our experiments. We attribute this to the absence of true labels from these duplicates.

Figure 7: Top: Near duplicates found in unlabeled images. Bottom: ImageNet-1k validation images.

D More Training Details

This section gives more training details on ImageNet classification, COCO detection, and ADE20K segmentation.

D.1 Intermediate Supervision Settings

We add intermediate supervision in ImageNet-1k training, ImageNet-22k and extra data pre-training. We used a 3-block decoder with gradually up-sampled feature maps in ImageNet-1k training. The block setting remains the same as Sec. 2.2. We use a single layer decoder in ImageNet-22k and extra data pre-training. For all the variants of RevCol, we set the number of compound loss n to 3 empirically (eg. for a 8 column RevCol, the intermediate supervision is added to column 2, 4, and 6, and the original classification CE loss is also added to column 8). α_i is set to 3, 2, 1, 0 and β_i is set to 0.18, 0.35, 0.53, 1.

D.2 Hyperparameters Used for Training and Pre-training

This section introduces the training details for main experiments, the supervised training on ImageNet and extra data. We show this setting in Tab. 11. All experiments in ablation studies are superivised trained on ImageNet-1K except additional descriptions and also follow settings described in this section.

Table 11: Hyperparameters for training and pre-training RevCol.

Hyperparameters	ImageNet-1K		ImageNet-22K	168M Extra Data
	T/S/B		B/L/XL	XL/H
Input resolution	224^2			224^2
Training epochs	300		90	10
Warmup epochs	20		5	0.15
Batch size	4096			5120
Peak learning rate	4e-3		5e-4	6.25e-4
Learning rate schedule	cosine			cosine
Layer-wise learning rate decay	✗			✗
AdamW momentum	(0.9, 0.999)			(0.9, 0.999)
Weight decay	0.05		0.1	0.05
Gradient clipping	✗			1.0 (element-wise)
Drop path	0.1/0.3/0.4		0.3	0.2
EMA	0.9999		✗	✗
Label smoothing ε	0.1			0.1
Data augment	RandAug (9, 0.5)			RandAug (9, 0.5)
Mixup	0.8			✗
CutMix	1.0			✗
Random erase	0.25			✗

D.3 Hyperparameters Used for Fine-tuning

This section gives the hyperparameters used for fine-tuning on ImageNet-1K and downstrea COCO object detection and instance segmentation, ADE20K semantic segmentation tasks, as shown in Tab. 12, Tab. 13 and Tab. 14.

Table 12: Hyperparameters for fine-tuning RevCol on ImageNet-1K classification.

Hyperparameters	ImageNet-1K
	B/L/XL/H
Input resolution	384^2/384^2/384^2/640^2
Fine-tuning epochs	30
Warmup epochs	0
Batch size	512
Peak learning rate	5e-5
Layer-wise learning rate decay	0.9/0.8/0.8/0.8
AdamW momentum	(0.9, 0.999)
Weight decay	1e-8
Learning rate schedule	cosine
Head init scale	0.001
Drop path	0.2/0.3/0.4/0.5
EMA	✗/✗/✗/0.9999
Gradient clipping	10.0 (norm)
Label smoothing ε	0.1
Data augment	RandAug (9, 0.5)
Mixup	✗
CutMix	✗
Random erase	0.25

Table 13: Hyperparameters for fine-tuning RevCol on object detection with Cascade Mask R-CNN detector.

Hyperparameters	IN-1K Pre-trained	IN-22K Pre-trained
	RevCol-T/S/B	RevCol-B/L
Fine-tuning epochs	36	
Batch size	16	
Peak learning rate	2e-4	1e-4
Warmup steps	1500	
Layer-wise learning rate decay	0.85/0.8/0.8	0.9/0.8
AdamW momentum	(0.9, 0.999)	
Weight decay	0.05	
Drop path	0.3/0.4/0.4	0.5/0.6

Table 14: Hyperparameters for fine-tuning RevCol on ADE20K semantic segmentation with UperNet segmentation framework.

Hyperparameters	IN-1K Pre-trained	IN-22K Pre-trained
	RevCol-T/S/B	RevCol-B/L
Input resolution	512^2	640^2
Fine-tuning steps	80k	
Batch size	16	
Peak learning rate	4e-5	
Warmup steps	1500	
Layer-wise learning rate decay	1.0	0.9
AdamW momentum	(0.9, 0.999)	
Weight decay	0.01	
Drop path	0.3	

D.3.1 Convolution kernel padding trick in down-stream tasks

According the results shown in Section 3.5.5, larger kernel convolution perform better especially in down-stream tasks. To save the pre-training cost meanwhile achieve better performance, we pad the small 3×3 convolution kernel in pre-trained model weights to larger size then fine-tune in detection and segmentation tasks. Inspired by *Net2net* (Chen et al., 2015) method, we pad the pre-trained 3×3 kernel in convolution layer with Gaussian initialized values. To protect the pre-trained kernel from being disturbed by the new padded values, we initialize the padded values with 0 mean and extremely small standard deviations (1e-7). We use this trick only with our largest model RevCol-H. We pad the 3×3 kernel in pre-trained model to 7×7 kernel size in COCO detection task and 13×13 in ADE20k sementatic segmentation task, then fine-tune on corresponding dataset to get the final result. In general, the kernel padding trick leads to 0.5$\sim$0.8 AP_{box} improvement and 0.7$\sim$1.0 mIoU improvement for RevCol-H model.

E Visualizations of Feature Disentanglement

In this section, we show our RevCol can disentangle features with stacked columns, which is different from the conventional sequential networks. We use RevCol-S pre-trained on ImageNet-1K for analysis. First, we visualize the class activation maps (CAMs) for outputs of each last layer of a level. We adopt LayerCAM (Jiang et al., 2021) technology to generate the CAMs with the predicted classes. Fig. 8 show the heatmaps of activation. With the levels and columns going deeper, the features focus on the regions with more semantics. The outputs of RevCol-S are the different levels of last column. These features with high level semantics focus on different parts of the image and the whole part of the object, achieving disentanglement of features for task-relevant and task-irrelevant.

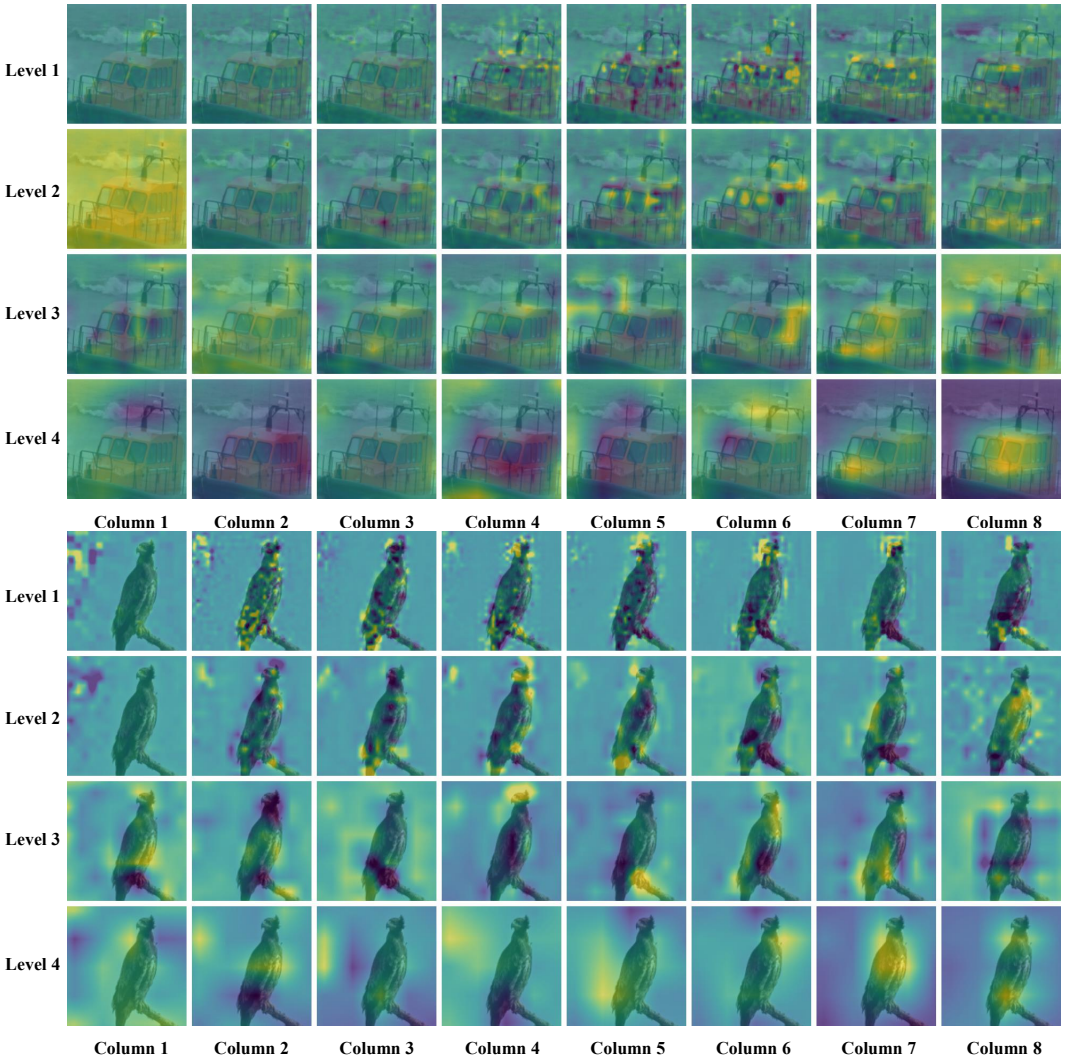


Figure 8: Visualizations of class activation maps using LayerCAM (Jiang et al. 2021) for different levels and columns.

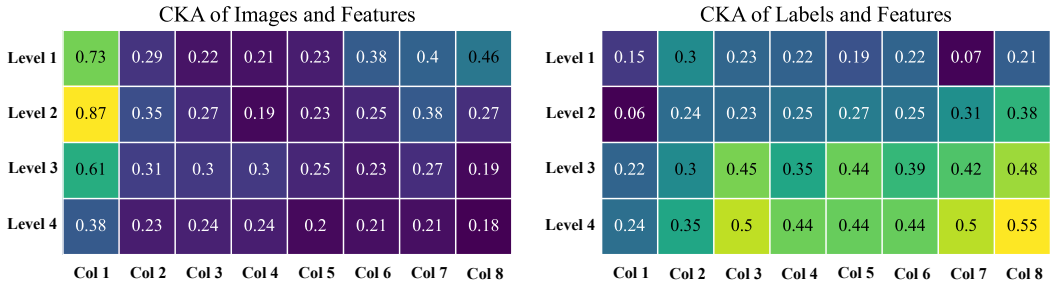


Figure 9: CKA similarities (Kornblith et al. 2019) of features and images/labels for different levels and columns.

To quantify the disentanglement, we use Centered Kernel Alignment (CKA) similarity metric (Kornblith et al. 2019) to measure the similarity between representations in RevCol-S. We calculate the CKA similarities between intermediate features in different levels and columns and images or labels of each category in the ImageNet val set. Then we plot the similarities of the category with

the highest label similarity in Fig. 9. As shown in the figure, the similarities between images and intermediate features are not clearly distinguished at different levels in Column 2-5, while the features with higher levels have lower similarity to the images in Column 6-8. The similarities between labels and intermediate features are also more distinct in higher columns.