MLP Coursework 3: Motivating Capsule Network Architectures

G63: s1310324, s1779598, s1164250

Abstract

Capsule networks (CapsNet) are a novel type of deep neural network first proposed in late 2017 (Sabour et al., 2017). Since they are such a recent development, only a small number of simple CapsNet architectures have been implemented and tested thus far. In this interim report, we outline a number of proposed experiments to explore the relative efficacy of different CapsNet architectures. We also implement and report on initial experiments which will serve as baselines for future work. Specifically, we trained a convolutional neural network (ConvNet) on the CIFAR 10, Small NORB and EMNIST data sets, and achieved a classification accuracy on the validation set of 76.520%, 77.013%, and 87.652% respectively. Moreover, we implemented a 3-layer CapsNet, and found a classification accuracy on the validation set of 63.777%, 91.629%, and a preliminary result of 87.449% respectively. Our subsequent experiments aim to improve upon this by investigating the effect of neural network depth on classification accuracy. In addition to this, we intend to vary the number of capsules per layer and assess the effects on performance. We also outline a strategy for the implementation of fractal architectures in CapsNets to assess whether these structures offer the same advantages in CapsNets as they do in traditional ConvNets. Finally, we aim to demonstrate the advantages of CapsNets over traditional ConvNets by investigating their relative performance in cases where pose and translational invariance is important for classification, for example on the Small NORB dataset.

1. Introduction and motivation

In recent years ConvNets have achieved or exceeded human-level accuracy on numerous image classification tasks, such as MNIST (LeCun et al., 1998), CIFAR 10 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009). However, despite their success, there remain fundamental limitations to traditional ConvNets, which make them unsuitable for certain applications without prior modification.

1.1. The Limitations of Convolutional Networks

Typically, a ConvNet is comprised of multiple alternating convolutional layers and max-pooling layers (LeCun et al., 1998). In a convolution, a stack of feature maps W of dimension $m \times m \times l$ is convolved with the output M from the previous convolutional layer, to give the output N of the next layer. The output of each convolution is an array of sums of element-wise products between feature map components and input elements at different regions of the input array:

$$N_{ijk} = \sum_{\tau=0}^{l} \sum_{\mu=0}^{m-1} \sum_{\nu=0}^{m-1} W_{\mu\nu k} \times M_{i+\mu, j+\nu, \tau} + b. \qquad (1)$$

Here b is a shared bias. These convolutions are designed to detect distinct features within particular sections of the image. The subsequent step is a pooling process which reduces the dimensionality of the stack of 2D arrays. Max-pooling is one popular option, in which, generally, an $n \times n$ mask moves across the image with a stride length equal to its dimension, and selects the largest output in each $n \times n$ patch of the output from the preceding convolutional layer (LeCun et al., 1998). The dimensions of the subsequent layer are therefore reduced by a factor of n.

This pooling operation means that ConvNets are largely incapable of discriminating between objects in different poses and positions (Hinton et al., 2011). Consider the process by which an object is classified at test time. In the first convolutional stage, a series of feature maps are convolved with different regions of the image: the higher the output of that operation, the greater the prevalence of the feature in that region (Lee et al., 2009). In the subsequent max-pooling stage, the highest value is selected from an $n \times n$ region, and is passed to an image of reduced area. The output of the max-pooling layer is therefore independent of small translational variations in the location of the feature, and information about the location of the feature is lost (Lee et al., 2009; LeCun et al., 1998). Over several layers, this effect is compounded and the spatial resolution of feature maps becomes increasingly course-grained; by the end of the feed-forward process there is little residual information about the relative location of features (Hinton et al., 2011). Therefore, ConvNets are a largely translationally invariant classification process (LeCun et al., 1998; Lee et al., 2009).

As a consequence, if we wish to classify an image (e.g. horse), then the mere presence of certain features (e.g. legs, head, tail) may be sufficient to result in a classification, without any concern for the precise positioning of those features (Hinton et al., 2011). A distorted image containing a rearrangement of those body parts might well be classified as a horse by the network, without any distinction being

drawn between the two cases.

Good image classifiers must be able to classify an object regardless of where and how it is positioned within an image (LeCun et al., 1998). However, the failure of ConvNets to differentiate between objects in different poses is unsatisfactory; an ideal system would be simultaneously able to classify whether or not an object was present in an image without losing any information about the location of this object or its pose.

1.2. Capsule-Based Architectures

CapsNets are an attempt to address problems with traditional convolutional architectures by negating the necessity for max-pooling (Sabour et al., 2017). To detect complex, large-scale features, detectors must receive inputs from an increasingly large section of the image as we propagate through the network (Lee et al., 2009). The challenge is to achieve this without resorting to max-pooling and losing information about the location of features.

To understand how CapsNets achieve this, it is helpful to outline a mathematical description of the behaviour of a capsule layer. The output of a single layer of a CapsNet is composed of N_1 capsules, each of which contains $m_1 \times m_1$ output vectors. Every such vector is of dimension l_1, and is denoted $\vec{u}_i$, with $i \in \mathbb{N}, i \leq (N_1 \times m_1 \times m_1)$. Here, we maintain the notation used in Sabour et al. (2017). The first step of processing in the subsequent layer is to define prediction vectors $\hat{u}_{i|j}$ as (Sabour et al., 2017):

$$\hat{u}_{i|j} = W_{ij}\vec{u}_i, \tag{2}$$

where each W_{ij} is an $l_1 \times l_2$ weight matrix. The j index enumerates the capsule vectors in the next layer. We subsequently define a new quantity as a linear combination of these vectors (Sabour et al., 2017):

$$\vec{s}_j = \sum_i c_{ij}\hat{u}_{i|j}. \tag{3}$$

The coefficients c_{ij} (referred to as coupling coefficients), are defined as a weighted sum of exponents of coefficients b_{ij}:

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ij})} \tag{4}$$

Intuitively, the coupling coefficients may be regarded as the probability that i and j are connected (Sabour et al., 2017). This step combines the dimensionality reduction advantages of max pooling without explicitly throwing out information about feature location. The resulting vectors $\vec{s}_j$ then serve as inputs to a non-linear squashing transformation:

$$\vec{v}_j = \frac{\|\vec{s}_j\|}{1 + \|\vec{s}_j\|^2} \times \vec{s}_j. \tag{5}$$

This function can be thought of as analogous to the non-linear element-wise activation function (e.g. ReLU) in a traditional feed-forward neural network, only now, instead of being an element-wise function, the non-linearity is dependent on a vector output (Sabour et al., 2017). Also, this guarantees that the magnitude of the vector is constrained to lie in the interval $[0, 1]$.

The coupling coefficients are continually updated via a routing algorithm (Sabour et al., 2017):

$$b_{ij} \leftarrow b_{ij} + \hat{u}_{i|j}.\vec{v}_j. \tag{6}$$

For an individual j, the coupling coefficient c_{ij} that will increase by the most in a single update is the one corresponding to the prediction vector $u_{i|j}$ for which $\hat{u}_{i|j}.\vec{v}_j$ is largest. The coupling coefficients determine how much each possible parent contributes to the vector v_j. Each v_j has the capacity to include contributions from all regions of the image, but it is likely that after repeated weight updates, it will become primarily dependent on a few parents. Nonetheless, unlike with max-pooling there is no explicit loss of information about the feature location. We hypothesise that this will ensure CapsNets are more adept at encoding discrepancies in pose and orientation than ConvNets. Initial experiments support the claim that CapsNets have a greater ability than traditional ConvNets to accurately classify images subject to affine transformations (Hinton et al., 2018; Sabour et al., 2017).

Sabour et al. (2017) also define the loss function for CapsNets as the combination of a margin loss and a reconstruction loss. The margin loss is given per capsule for class k as

$$L_k = T_k \max(0, m^+ - \|\vec{v}_k\|)^2 + \lambda(1 - T_k)\max(0, \|\vec{v}_k\| - m^-)^2 \tag{7}$$

where $T_k = 1$ if class k is present, λ scales down the penalty for absent classes, and m^+ and m^- are constants. The total loss is the sum of L_k for all k and a scaled-down reconstruction loss given by the sum of squared difference of the original image and reconstruction. The reconstructions, such as those in Figure 2, are the result of using several fully-connected layers to decode the last capsules' output masked by either the prediction or target.

1.3. Research Questions

The central claim of (Sabour et al., 2017) is that CapsNets are capable of accounting for variations in pose and position. We aim to assess the validity of this assertion by comparing the performance of CapsNets and ConvNets on data sets where there is significant variation in pose and orientation of objects in the image. Moreover, we may apply CapsNets to the more involved task of predicting object orientation from data, by attempting a variation of the regression task carried out in (Hara et al., 2017). This would give a clear indication of whether CapsNets are capable of quantitatively estimating variations in pose.

The architectures implemented in CapsNets thus far have been fairly simple and comprised a small number of layers (Sabour et al., 2017). We wish to explore how changes in the architecture of capsules affect their performance. Our initial plans include investigating the effects of network

depth and the number of capsules on classification accuracy, since these changes are fairly simple to implement. In each case the CapsNet will be compared to traditional ConvNets with similar architectures. The implementation of non-trivial architectures has led to significant advances in ConvNet performance in recent years (He et al., 2016; Larsson et al., 2017). We aim to discover whether skip connections and fractal architectures in CapsNets offer the same benefits as they do with more traditional network architectures.

2. Objectives

The results from the first implementations of a CapsNet were only released in late 2017, so there remains a lack of evidence in relation to the comparative efficacy of different capsule-based architectures. In this project, we aim to study the performance of CapsNets with differing structures by comparing their accuracy on image classification tasks. At every stage, we anticipate comparing CapsNets to vanilla ConvNets with comparable numbers of parameters, in order to assess whether CapsNets offer improved performance over established techniques.

2.1. Baselines

Our first task is to implement a number of baselines, using both CapsNets and traditional ConvNets. Our baseline CapsNet architecture is similar in structure to the one implemented in (Sabour et al., 2017). For comparative purposes, we will also implement a ConvNet with a similar number of parameters to assess performance variations. Subsequent experiments will aim to improve upon these baselines.

2.2. Variations in Capsule Number

An individual layer in a standard ConvNet is simply an array of hidden units. What distinguishes CapsNets is that this array of units is now further subdivided into capsules, and each such capsule contains several vectors of hidden units. Therefore, it is natural to establish how changing the number of capsules and the dimension of these vectors affects the performance of the neural network. Architectural changes of this nature are simple to implement, so this is a good initial avenue of inquiry.

2.3. Exploring Deeper CapsNet Architectures

Thus far, the CapsNets implemented in the literature have been relatively shallow, comprising no more than three or four layers (Sabour et al., 2017; Hinton et al., 2018). We aim to explore how deeper capsule architectures perform on our two data sets (CIFAR 10 & Small NORB). Clearly, deeper networks require more data for training, so it is expected there will come a point at which the networks begin to over-fit for a fixed quantity of data. However, deeper networks are able to approximate more complex functions than shallow networks of an equivalent width (Goodfellow et al., 2016), and very deep architectures have previously led to state-of-the-art performance on classification tasks (He et al., 2016). For these experiments, we anticipate maintaining simple structures composed of consecutive processing layers devoid of skip connections.

2.4. Implementing Complex Architectures

In recent years, neural network architectures have taken on increasingly elaborate forms, and this has led to significant increases in classification accuracy (Larsson et al., 2017; He et al., 2016). Current implementations of CapsNets rely on simple architectures in which consecutive layers are connected only to each other (Sabour et al., 2017). We wish to apply modern architectural developments to CapsNets, and assess whether these structural changes lead to similar improvements in performance.

Initially, we plan to implement a fractal network of convolutional capsule layers. Fractal networks are a sophisticated neural network architecture: the input to a single layer is a composition of the output of multiple distinct sub-networks of varying depths (Larsson et al., 2017). Mathematically, for some given input z, the output of a fractal network of order C is defined as (Larsson et al., 2017):

$$f_C(z) = (f_{C-1} \otimes f_{C-1})(z) \oplus f_1(z). \tag{8}$$

Here, the $\otimes$ operation denotes a composition of two separate operations and the $\oplus$ operation combines the layers by taking the element-wise weighted average of the outputs. This recursive definition means that an $f_2(z)$ fractal network would be some weighted average of a single layer network $f_1(z)$ and the composition of two further $f_1(z)$ layers. In our case, we intend to implement a network where the fundamental layer is a convolutional capsule layer, i.e.:

$$f_1(z) = ConvCaps(z) \tag{9}$$

If time permits, we may explore additional modern architectures. Some of the best performing image classification systems are residual networks (or ResNets) (He et al., 2016). These architectures contain skip connections, in which a single layer l takes as its inputs the outputs from both layers $l-2$ and $l-1$ if l is even (He et al., 2016). We may look at implementing these in a CapsNet if time permits.

2.5. Azimuth Estimation via Multi-Task Learning

As we have already mentioned, perhaps the most exciting aspect of CapsNets is their perceived ability to encode features such as angle and pose. In order to explore this property further, we intend to implement a multi-task learning objective, which simultaneously learns to classify a Small NORB image and estimate the azimuthal angle at which the image was taken. If CapsNets display significant performance improvements relative to vanilla ConvNets on this task, then this will be a very strong indication of their superior ability to learn pose.

This is an ambitious task, and it will only be attempted if there is sufficient time and all other experiments are completed. As a simpler alternative, we may forgo the

classification task and focus solely on training the network to predict the azimuthal angle via regression from a series of images.

3. Dataset and task

We intend to compare the relative performance of CapsNets and vanilla ConvNets on the CIFAR 10 and Small NORB data sets. In both cases, we are performing a classification task, where a single class label is assigned to a particular image. We aim to maximise classification accuracy on a validation set, and evaluate the best model on a held-out test set.

CIFAR 10 is a data set containing 60,000 colour images, each of dimension 32×32, with 6,000 images allocated to 10 distinct classes (Krizhevsky et al., 2009). Of these images, 1,000 per class are held out for testing and 1,000 are used as validation (Krizhevsky et al., 2009). It should be noted at this point that existing experiments on CIFAR found that CapsNets markedly under-perform current state-of-the-art ConvNets (Sabour et al., 2017). However, if CapsNets are to be applied successfully to real-world object classification or identification tasks, then improving their performance on real-world image sets should be a priority.

In contrast to CIFAR, Small NORB is a data set consisting of 96×96 pixel grey scale images of small toys, viewed from different angles, elevations and azimuths (Figure 1) (LeCun & Huang, 2004). Small NORB is a subset of the larger NORB data set, and consists of 24,300 training images, and a further 24,300 held out images (LeCun & Huang, 2004). There are 5 different categories of toy, and 10 separate examples within each category, of which 5 are allocated to the training set and 5 to the held-out set. Furthermore, each example is viewed from 18 different azimuths, 9 elevations and 6 distinct illuminations (LeCun & Huang, 2004). The initial task here is to classify each image by class. Like Sabour et al. (2017), we downsampled the 96×96 images of the Small NORB dataset by discarding alternating columns and rows, resulting in images of size 48×48. We only considered examples from one of the two cameras. We use 9,720 held-out images of 2 toys from each category as a validation set.

So far, we have only discussed classification tasks, but each Small NORB image comes with a set of labels specifying the azimuth, elevation and lighting angle. If time permits, we intend to perform a multi-task learning assignment, in which the CapsNet must simultaneously predict both the class and the azimuth. The azimuth is a continuous variable, so this component of the task would be involve regression rather than classification.

Finally, we also conducted initial experiments on the EMNIST Balanced dataset (Cohen et al., 2017), as in coursework 2. EMNIST Balanced is a dataset containing 131,000 greyscale 28×28 images of handwritten numbers and letters. The dataset merges upper and lower-case C, I, J, K, L, M, O, P, S, U, V, W, X, Y, and Z, so there are 47 different classes. The data is split up as follows: 100,000 training images, 15,800 validation images, and 15,800 test images.

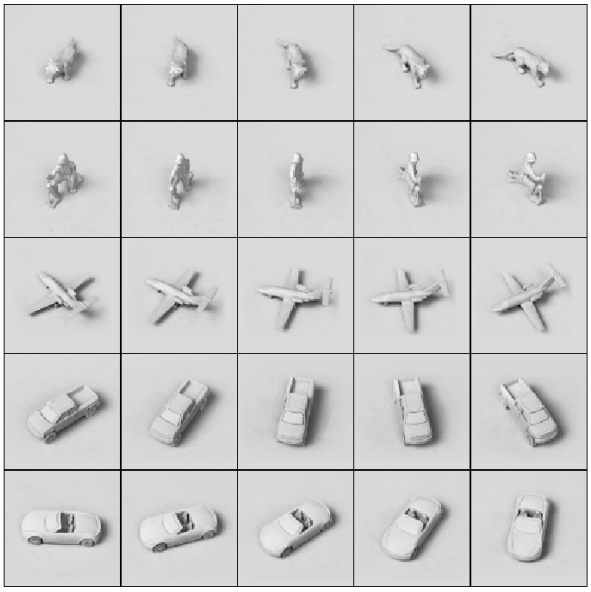


Figure 1. Examples from the Small NORB dataset. (LeCun & Huang, 2004) Each row contains five examples of an instance from each of the five categories. Each column presents the object from a different azimuth.

4. Methodology and Baseline experiments

The best ConvNet and CapsNet were trained on the three different datasets: EMNIST, CIFAR 10 and Small NORB. The CIFAR 10 architectures are shown in Table 1 and Table 2. For CIFAR 10, the CapsNet had 11.7M parameters, and the ConvNet had 11.5M. For Small NORB, the CapsNet had 14.0M parameters, and the ConvNet had 13.5M and fully-connected layers of size 300, 512, and 1,024. For EMNIST, the CapsNet had 14.0M parameters and ConvNet had 13.7M and fully-connected layers of size 576, 1,024, 1024.

4.1. A Framework for Reproducible Research

Ideally, it should be easy to associate experiment results with the full computational environment. To this aim, we designed and implemented a framework for managing experiments. In the framework, a YAML configuration file defines an experiment's network architecture, such as layer types, loss functions, and other hyperparameters, such as optimiser learning rates or early stopping criteria. These are translated into a TensorFlow graph. While the framework is original work, the CapsNet component is derived from Aurélien Geron's[1] and other features are derived from the

[1] https://github.com/ageron/handson-ml/blob/f9ee98018dc0346759a457112a8944d67e9b9c4f/extra_capsnets.ipynb

CIFAR10: Input $32 \times 32 \times 3$, 10 classes; batchsize 50
5×5 conv. 256 Leaky ReLU units. stride 1
batch norm
5×5 conv. 256 Leaky ReLU units. stride 1
batch norm
maxpooling 2×2
5×5 conv. 512 Leaky ReLU units. stride 1
batch norm
3×3 conv. 512 Leaky ReLU units. stride 1
batch norm
FC 200 Leaky ReLU units
FC 400 Leaky ReLU units
FC 1024 Sigmoid units
softmax

Table 1. ConvNet Model Architecture.

CIFAR 10: Input $32 \times 32 \times 3$, 10 classes; batchsize 50
9×9 conv. 256 Leaky ReLU units. stride 1
9×9 primary capsule. 32 capsules 8; stride 2; Leaky ReLU
digit capsule: 10 capsules of dim 16.
dynamic routing: 2 iterations.
(prediction)
mask
FC 512 Leaky ReLU units
FC 1024 Leaky ReLU units
FC 3072 Sigmoid units

Table 2. CapsNet Model Architecture.

MLP Practical codebase.[2] This framework can be extended. For example, in the future it could choose hyperparameters for new configurations based on the results of previous experiments using Bayesian optimization.

4.2. Baseline Convolutional Neural Network

In order to measure the performance of our CapsNet, we trained ConvNets on CIFAR 10 and Small NORB, comparing different architectures and picking the ones that performed best as our baselines.

Unless otherwise stated, we used the Adam Optimizer (Kingma & Ba, 2015) with a learning rate of 0.001 and $\beta = 0.9$.

In order for ConvNets results to be comparable to our CapsNet results, we needed them to have a similar number of parameters, and so we chose to restrict the depth of our ConvNet to 4-6 layers. After some experimentation, we settled on the following architecture: 2 convolutional layers, a max-pooling layer, followed by 2 convolutional layers, 3 fully connected layers, and finally a softmax layer for classification. We used batchnorm between each convolutional layer and a learning rate η of 0.0001. The details of the architecture can be found in Table 1. Using this architecture, the ConvNet achieved a validation accuracy of 76.520%.

[2] https://github.com/CSTR-Edinburgh/mlpractical/tree/8ac193d9065e957fc6103dc7eca314b6c7844696

We then varied the activation functions, the learning rate, and the lengths of the fully connected layers at the end, and compared the changes to the activation function relative to the original setup, with respect to validation accuracy. Our results are summarised in Table 3. Changing the activation from leaky ReLU to ReLU and adding more neurons to the fully connected layers did not significantly affect performance. However, increasing the learning rate substantially decreased validation accuracy. This led us to further experimenting with the learning rate, and we concluded that $\eta = 0.0001$ was optimal.

Parameters	Original	Deviation	Difference
FC Layers	200/400	700/1400	+0.01%
Activation Fns	Leaky ReLU	ReLU	-0.4%
Learning Rate	0.0001	0.001	-41.267%

Table 3. Difference in validation accuracy on CIFAR 10 from initial architecture relative to the number of neurons in the fully connected layers, the activation functions, and the learning rate.

We also tried replacing the max-pooling layer with convolutions of stride 2, as in Springenberg et al. (2015). However, we did not find that this made much difference. Furthermore, by randomly rotating some input images during training, we improved our best validation accuracy on CIFAR10 from 76.520% to 77.360%. However due to time limitations we weren't able to run comparable models with a CapsNet or the other datasets.

4.3. CapsNet Baseline

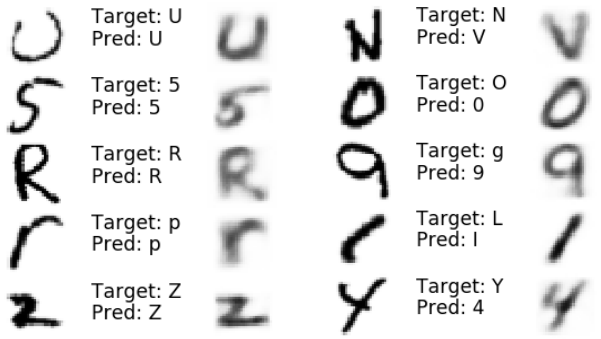
In general, our CapsNets are implemented as described in subsection 1.2. The specific baseline architecture is in Table 2. Like Sabour et al. (2017), we scaled the reconstruction loss using $\alpha = 0.0005$, and use $m^+ = 0.9$, $m^- = 0.1$, and $\lambda = 0.5$ in the margin loss.

We saw an improvement using Leaky ReLU instead of ReLU. On CIFAR, the Leaky ReLU network achieved a validation accuracy of 63.773% compared with the ReLU-network's 58.947% (cf Table 5). Using no activations on the Small NORB dataset required a lower learning rate in order to train.

The much higher number of classes of EMNIST introduced a computational challenge. A single epoch took 2 hours to complete, and we were forced to stop the run before it reached its optimum.

5. Interim conclusions

From our initial baseline experiments, it is apparent that CapsNets offer a slight improvement over ConvNets on the Small NORB classification task. These findings are consistent with the preliminary results from existing academic literature (Sabour et al., 2017). We attribute this in part to the ability of CapsNets to encode changes in pose. It should be noted that the original implementation augmented the



(a) Correctly classified (b) Incorrectly classified

Figure 2. Example reconstructions from our EMNIST CapsNet model. Both figures show inputs on the left, targets and predictions in the center, and CapsNet's reconstructions on the right. The impact of the incorrect predictions on the decoded output is evident in the right figure.

Arch/Dataset	CIFAR 10	Small NORB	EMNIST
ConvNet	**76.520%**	90.649%	87.652%
CapsNet	63.773%	**91.629%**	87.449%*

Table 4. The results of our best ConvNet and CapsNet architectures on CIFAR 10, small NORB, and EMNIST. *Due to time constraints we did not use early stopping on EMNIST.

data and classified using a series of mixture models, so their results were superior to those reported here (Sabour et al., 2017).

However, CapsNets performed worse than ConvNets with a comparable number of parameters on the CIFAR 10 dataset; this is again in keeping with the findings of (Sabour et al., 2017). The authors attribute this to significant variation in the background of CIFAR images: CapsNets attempt to account for all features in an image, including those not important for classification (Sabour et al., 2017). In view of this, they perform better if they can "explain away" the background. For this reason the authors elected to include an "other" category, to which all features not belonging to the set of classes could be allocated (Sabour et al., 2017). It might be interesting to investigate at a later date whether introducing such classes into our code could improve performance. It should be noted that in the Small NORB data set, the backgrounds are largely homogeneous, so this is less of an issue.

Dataset	CIFAR10	Small NORB
None	55.227%	90.010%
ReLU	58.947%	90.021%
Leaky ReLU	**63.773%**	**91.629%**

Table 5. The effect of activation functions on CapsNet validation accuracy on the CIFAR 10 and Small NORB dataset. *None* indicates no activation functions on the convolutions and ReLU on fully-connected layers. The others indicate that activation function on both the convolutions and fully-connected layers.

These initial results are promising, and suggest that our initial plans to further explore the impact of architectural changes on CapsNets are warranted, especially in view of the lack of currently available evidence in the literature. Moreover, our final objective, which involves using CapsNets for a regression task to predict orientation, will give a clearer insight into the extent to which CapsNets are capable of identifying variations in pose, which is the central claim of (Sabour et al., 2017).

6. Project Plan

Following our initial experiments, there are several lines of inquiry which we will explore further.

We will first explore the effects of basic architectural changes on the performance of CapsNets. This would involve adding layers to the network and varying the number of capsules within each layer. Such modifications are simple to implement, and will give us insight into the behaviour of CapsNets.

Once these experiments are running, the next stage will involve implementing a fractal network with capsules and/or introducing skip connections to our best performing current network. These tasks present greater challenges from a coding perspective; however, at present our networks have been set up such that it is easy for any given layer to take inputs from multiple prior layers, and therefore we believe this implementation to be feasible within the allocated timeframe.

Finally, time-permitting we will move on to our most ambitious objective: a multi-task learning assignment on Small NORB in which the CapsNet must simultaneously predict both the image class and angle. If it becomes apparent that this will take too long to implement, we may instead attempt a simpler regression assignment, where the network only learns the angle without predicting the class.

At this moment in time, we have no reason to believe that any individual objective is not achievable. However, the average training time for a typical capsule layer with our current code is 20 minutes, and this will increase as we explore deeper architectures. This is an order of magnitude higher than the training time of a vanilla ConvNet with a comparable number of parameters. Therefore, we run the risk that time constraints may prevent us from conducting all of the experiments proposed here. Also, it is worth noting that both fractal networks and CapsNets are extremely new innovations in deep learning[3]. As of the time of writing, it is very unclear how capsule structures will perform in a network with a fractal architecture.

[3]Fractal networks are less than two years old. CapsNets were first implemented less than four months ago! (Larsson et al., 2017; Sabour et al., 2017)

References

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hara, K., Vemulapalli, R., and Chellappa, R. Designing deep convolutional neural networks for continuous object orientation estimation. In *Conference on Computer Vision and Pattern Recognition*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.

Hinton, G., Krizhevsky, A., and Wang, S.D. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pp. 44–51. Springer, 2011.

Hinton, G., Sabour, S., and Frosst, N. Matrix capsules with em routing (under review). In *International Conference on Learning Representations*, 2018.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. URL http://arxiv.org/abs/1412.6980.

Krizhevsky, Alex, Nair, Vinod, and Hinton, Geoffrey. Cifar-10 (canadian institute for advanced research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

Larsson, G., Maire, M., and Shakhnarovich, G. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations*, 2017.

LeCun, Y. and Huang, F. Learning methods for generic object recognition with invariance to pose and lighting. 2004.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning*, 2009.

Sabour, S., Frosst, N., and Hinton, G.E. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3859–3869, 2017.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin A. Striving for simplicity: The all convolutional net. 2015. URL http://arxiv.org/abs/1412.6806.