MLP Coursework 4: Investigating the Properties of Capsule Networks

G63: s1310324, s1779598, s1164250

Abstract

In this report, we explore the properties of Capsule Networks (CapsNets), a novel type of convolutional neural network proposed last year. Our report builds on the work of Sabour et al. (2017) and explores how various modifications to their original architecture affect the network's performance. We consider the relative effectiveness of a number of different capsule network architectures for the task of image classification. Throughout this report, we use a course-grained version of the small NORB data set of toy images for all our experiments (LeCun & Huang, 2004).

We found that altering the length of the vectors in the primary capsule produced little change in the performance of the network, though we note that networks with longer feature vectors perform marginally better and exhibit superior compute times. Increasing the depth of the network by adding more convolutional layers before the capsule layers also failed to improve the performance. We then consider a network which contained a skip connection over the first capsule layer; we found that including this skip resulted in a marginal improvement in performance.

In the final sections of the report, we move on to consider a multi-task learning assignment in which the network must learn both an image's class and its angle of orientation. Combining the two tasks did not seem to produce any particular improvement in performance.

Overall, we found that our best performing network contained a single skip connection over the primary capsule layer. The classification accuracy on a course-grained version of small NORB was 89.4% ± 0.3%.

1. Introduction

In this project, we investigate the properties of capsule networks, a novel type of deep architecture, introduced by Sabour et al. (2017). Capsule networks were developed as a method of better accounting for pose variations in convolutional architectures. As discussed in the previous report (G63, 2018), the max-pooling operation, which is frequently used in convolutional architectures, introduces a degree of translational invariance (Lee et al., 2009). Successive layers of max-pooling result in a loss of information about the relative location of the features associated with a particular object (Hinton et al., 2011). This can be problematic when dealing with a data set containing objects with significant variation in pose, since the relative location of an image's features is important for estimating object pose.

Capsule networks avoid max-pooling through a routing procedure, in which each node in layer l selects a particular vector or series of vectors in layer $l-1$ from which to receive its input (Sabour et al., 2017). Each vector is associated with a feature in a particular region of the image. Therefore, a single feature vector in layer l consists of a series of nodes which are in general a measure of the strength of a specific feature at a particular location in the previous layer $l-1$. Crucially, the vectors in layer l can select their input from any feature vector in layer $l-1$, from any part of the image. Consequently, each feature vector in layer $l-1$ 'sees' the entire scope of the image. Therefore, the network is capable of learning highly complicated features which span the whole of the image, but without resorting to explicitly transitionally invariant down-sampling methods such as max-pooling (Sabour et al., 2017). A more detailed discussion of this can be found in the previous report (G63, 2018).

Our objectives in this project are two-fold. Firstly, we make a number of modifications to the original architecture in an attempt to improve the performance of the network. The architecture proposed by Sabour et al. (2017) is relatively simple, comprising only 3 hidden layers, and therefore, it seems plausible that simple modifications to the architecture might lead to improved performance on image classification tasks. Secondly, we also aim to gain an understanding of how pose variations are encoded in capsule networks.

Initially, we investigate how changing the length of the vectors in a capsule network affects the performance of the network. Each individual vector is regarded as a representation of a particular image feature. Assuming the number of parameters is held constant, there is a trade-off between the number of features in any given layer and the complexity of those features, which is assumed to increase with the length of the vector (Sabour et al., 2017).

Next, we move on to consider whether the addition of more convolutional layers improves the performance of the network. Deeper networks tend to be better universal approximators than shallower networks (Goodfellow et al., 2016), so it is reasonably intuitive to assume that deeper networks may result in better classification performance.

Our next modification was to introduce a network which contained a skip connection. In our objectives section in the previous report, we mentioned that our original plan was to implement a fractal network of capsule layers. However, it quickly became apparent that this network would be prohibitively expensive to train in terms of compute time, and we elected instead to implement a simpler network with a single skip connection over the first capsule layer. Computational expense was identified as a major risk in our first report, and a network involving skip connections seemed a sensible and less computationally expensive alternative to fractal networks. Previous incarnations of deep networks with skip connections have led to state-of-the-art performance in image recognition tasks (He et al., 2016). Consequently, we were interested to see if the use of skip connections in capsule networks would lead to similar improvements.

In order to further understand how the CapsNet represents images, we first explored the reconstruction phase of the CapsNet, which tries to recreate the input image based on the routing output layer. The reconstruction loss is subsequently used as one component of the CapsNet loss function. Specifically, we adjusted the weighting of the reconstruction loss and compared a CapsNet autoencoder to a CNN autoencoder.

The authors of (Sabour et al., 2017) suggest that capsule networks are better able to identify changes in object pose than convolutional neural networks. We therefore incorporate an additional loss function which measures the discrepancy between the true angle of the object in the image and that predicted by the capsule network. The idea is to establish whether this multi-task learning assignment, which effectively forces the network to account for variations in orientation, can outperform the regular network on classification assignments. We also investigate how different nodes in the routing output layer of the network correlate with different orientations of the image, to establish if a particular linear combination of nodes is responsible for encoding pose variation.

2. Methodology

We provide below a brief description of the architecture for the standard capsule network. All subsequent architectures we use in this report are a modification to the original architecture used in (Sabour et al., 2017), which we outline below.

The input to a capsule network is an image containing $n \times n$ pixels. The first layer is a normal convolutional layer, comprising 256 distinct 9×9 convolutional kernels. The resulting output is therefore of dimension $((n-8) \times (n-8) \times 256)$. We found in the previous report that leaky ReLU activation functions (Nair & Hinton, 2010) provided the best performance (G63, 2018), and so we continued to use these as the activation function for all convolutional layers in this report.

The next layer is referred to in Sabour et al. (2017) as the Primary Capsule layer, and for consistency we will use the same terminology here. This is a convolutional layer, with kernels of dimension 9×9 and a stride length of 2. However, the output is now subdivided into 32 separate capsules, each of which contains $(n-16) \times (n-16)$ vectors of length 8, labelled $\vec{s}_j$. Each vector represents a feature from a different region of the image. The activation function of this layer is a non-linear squashing function of the vectors:

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

The following layer is arguably the most important layer in the network and distinguishes capsule networks from normal neural networks. The output vectors are each transformed by a separate 16×8 weight matrix W_{ji}, to produce vectors $\vec{u}_{i|j}$.

$$\vec{u}_{i|j} = W_{ij}\vec{v}_j. \tag{2}$$

There are $32 \times (n-16)/2 \times (n-16)/2$ such weight matrices W_{ji}. The next layer is a referred to as the 'DigitCaps' layer, and consists of 5 capsules, each containing just a single vector of length 16. Each vector in this layer is a weighted sum of the $\vec{u}_{i|j}$ defined above.

$$\vec{s}_i = \sum_i c_{ji}\vec{u}_{i|j}. \tag{3}$$

These vectors are then subjected to the same non-linear squashing function as used in the output of the capsule layer. (equation 1)

The c_{ji} coefficients are updated via the routing procedure alluded to in the introduction. The change in the c_{ji} at every time-step is a function of the 'agreement' between the each of the 5 output vectors of the Digit Capsule layer $\vec{v}_i$ and the $\vec{u}_{i|j}$. The explicit computation is shown in the equation below:

$$c_{ji} \leftarrow c_{ji} + f(\vec{v}_i.\vec{u}_{i|j}). \tag{4}$$

The precise mechanism and form of the function f is discussed in more detail in the first report (G63, 2018). The main idea here is that by continually updating the coefficients c_{ji}, the DigitCaps layer can iteratively select which vector outputs of the Primary Capsule layer it takes as its input. Each individual vector in the DigitCaps layer is the activation vector for a particular class.

Our loss function is a combination of a margin loss and reconstruction loss. The margin loss is the standard softmax function of the class probabilities, whilst the reconstruction loss is a measure of the disparity between the reconstructed output and the original image. This reconstruction was achieved via a series of dense layers. The overall architecture is summarised in table 1.

tiny small NORB: Input $28 \times 28 \times 1$, 5 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: 5 capsules of dim 16.
dynamic routing: 2 iterations.
(prediction)
mask
FC 512 Leaky ReLU units
FC 1024 Leaky ReLU units
FC 784 Sigmoid units
Parameter count: 7.4M

Table 1. Architecture for CapsNet on tiny small NORB.

We used the small NORB data set in our analysis throughout our report, because the data set is explicitly constructed such that each class of image in the training set is photographed in a range of different orientations and scales(LeCun & Huang, 2004). This makes it an ideal data set for testing whether a classifier network is robust to changes in pose. The images in the data set are of a series of small toys, and these photographs are taken from a variety of different elevations and heights.

Unless otherwise stated, the following settings were used. Networks were trained using the Adam optimiser with a learning rate of 0.001 and $\beta = 0.9$. As a form of regularisation, models were early stopped after 11 iterations of no improvement in the validation set accuracy. The reported accuracy values are given by the average percentage of the images in the validation set for which the class is correctly predicted. If errors are given, the accuracy is the average of three runs and the error is the standard deviation. The activation functions for the ConvNets are leaky ReLUs (Nair & Hinton, 2010) by default. The default data set was small NORB. We downsampled the original 96×96 small NORB images in two ways: the 48×48 images in "small NORB" use every other row and column of the original, and the 28×28 images in "tiny small NORB" use every 3rd row and column of the original and then remove a border of width 2.

3. Experiments

In the following sections, we outline our experiments in detail and present our results and findings. As a comparative reference point, our baseline for small NORB from the first report was 91.6% (G63, 2018).

3.1. Additional hyperparameters

First, several experiments helped stabilise our baseline CapsNet on the 48×48 small NORB. By adjusting the learning rate up from 0.0001 to 0.001, we improved our validation accuracy from 91.6% up to **93.3**%.

Second we tried applying dropout between every layer in the CapsNet using a rate of 0.5. Dropout is a regularisation technique that can help cases where it fits too quickly to the training data (Srivastava et al., 2014). We got an accuracy of **92.3**%. While this didn't improve our baseline, there are many configurations in which dropout could be applied, and alternatives could be explored in future work.

tiny small NORB: Input $28 \times 28 \times 1$, 5 classes; batchsize 50
5×5 conv. 175 Leaky ReLU units. stride 1
batch norm
Dropout: 0.5
5×5 conv. 175 Leaky ReLU units. stride 1
batch norm
Dropout: 0.5
maxpooling 2×2
3×3 conv. 350 Leaky ReLU units. stride 1
batch norm
3×3 conv. 350 Leaky ReLU units. stride 1
batch norm
maxpooling 2×2
FC 450 Leaky ReLU units
FC 900 Leaky ReLU units
FC 2304 Sigmoid units
softmax
Parameter count: 6.3M

Table 2. CNN architecture for tiny small NORB.

One concern with the 48 small NORB data set was that the original models would take around 16 hours to train, and testing more complex models would increase this computational cost even more. One approach to improve the speed is to reduce the size of the input images. A way to do this is also a data augmentation technique used by Sabour et al. (2017) which crops the image to a random 32×32 area. This is particularly suitable for small NORB which has a large border and centred objects. At test time, the model classifies a centred-cropped image. This reached a classification accuracy of **88.3**%. An alternative solution is to forego data augmentation and crop and downsample all images, as we did with the *tiny small NORB* data set. Using a learning rate of 0.001, our CapsNet model in Table 1 achieved **89.3**% $\pm$ **0.2**%. We also compared this to a CNN shown in Table 2 on the same data set with learning rate of 0.0005, which achieved a similar **89.4**% $\pm$ **0.3**%, even with nearly a million fewer parameters. From these experiments, we found the learning rate made the biggest impact on validation accuracy. We also reported baselines on tiny small NORB, which will be compared in the deeper networks.

3.2. Adjusting Capsule Size and Number

Our initial experiments involved changing the length of the vectors in each capsule used for routing between adjacent layers of capsules. In Sabour et al's. original paper, the number of capsules in the primary capsule layer was set to 32, with each capsule containing vectors of length 8 (Sabour et al., 2017). The fundamental idea here is that there is a trade off between the complexity of the features and the number of features which can be encoded. If each vector in the Primary Capsule is considered the encoding of an image feature in a particular region, then longer vectors are

Vector Length	Capsules	Accuracy	Error
4	64	88.3%	1.2%
8	32	89.3%	0.2%
16	16	89.2%	0.3%

Table 3. A comparison of the relative efficacy for each of the different capsule networks on tiny small NORB. It is apparent that the original architecture proposed in (Sabour et al., 2017) performs marginally better than the modified architectures we propose here.

clearly capable of encoding more complex features (Sabour et al., 2017).

However, if the number of feature maps in the prior convolution is fixed at 256, then increasing the length of the vectors in each layer decreases the associated number of features which can be encoded. Moreover, increasing the number of capsules results in an increase in the number of weight matrices mapping outputs from the Primary Capsule layer to the Digit Capsule layer. We found this comes at a significant cost in terms of compute time.

To investigate how varying the vector length affected the performance of the capsule network, we compared 3 different networks, where the vector lengths were fixed at 4, 8 and 16 respectively. We trained each network on 4 separate occasions, with a different initialisation of the weights each time, on the tiny small NORB data set. Our results for the validation accuracies are shown in table 3.

The performances of all three networks are comparable, but the networks with vectors of length 16 and 8 do show a slight improvement over the network with vectors of length 4. Furthermore, it should be noted that the capsule network with the largest vector length requires significantly less time to train than the networks where the vectors are shorter, since there are fewer separate weight matrices mapping between consecutive layers. In view of this it is pleasing that the performance of the network with a vector length of 16 performance was comparable to that of the original network.

These results illustrate that changing the vector length does not have a huge impact on the network's performance, but using vectors which are fairly short (and therefore incapable of encoding complex features) may slightly compromise classification accuracy.

3.3. Deeper Architectures - Multiple Convolutional Layers

The use of extremely deep architectures in traditional convolutional networks has led to significant improvements in machine performance for image recognition tasks (He et al., 2016). Therefore it is natural to establish whether adding additional layers to our capsule network will lead to improved performance. We tested 3 different convolutional capsule networks of differing depths, again using the tiny small NORB data set to minimise compute time. The first was the original capsule network, which contained a single convolutional layer with 256 feature maps of size 9×9, stride 1, and a leaky ReLU activation function.

Configuration	Accuracy	Error
Network 1	89.3%	0.2%
Network 2	88.1%	0.8%
Network 3	88.7%	0.5%

Table 4. A comparison the performance of capsule networks with a different number of convolutional layers.

It was apparent that for deeper architectures it would be necessary to reduce the sizes of the kernels in the prior convolutional layers. Otherwise, the entire '2D plane' of the capsule, which spans different locations within the image, would have been represented by only a handful of units, and through successive convolutions the image would become overly course-grained. After all, part of the rationale behind capsule networks is to prevent the introduction of explicit translational invariance. Therefore it would be counter intuitive for the entire plane of the image to be represented by only (say) 2×2 different units, as there would be little residual information about the relative location of features.

With this in mind, we came up with two additional deeper architectures, with subtly different structures. Network 1 has the same architecture as that described in the methodology. In network 2, there is an additional convolutional layer (2 ordinary convolutional layers prior to the primary capsule layer), each of which have kernels of size 9×9. Now, however, the size of the kernel in the convolutional capsule network is reduced to 4×4 so that the output of the Primary Capsule layer is now of dimension 5×5. Finally, network 3 has 3 ordinary convolutional layers, but the kernel dimension is now reduced to 5×5. The size of the kernel in the Primary Capsule layer is again reduced to 4×4. Note that all convolutional layers (including the Primary Capsule layer) contain 256 feature maps.

Again, each network was run 4 different times with a different random initialisation of the weights. The performance of each network is summarised in table 4.

From the results above, it is apparent that adding additional convolutional layers prior to the capsule layer does not appear to significantly improve the performance of the architecture. This is somewhat surprising, as we anticipated a priori that a larger number of successive convolutions would allow the network to learn more complex features and thereby better identify images.

There are a couple of possible explanations for this. One is that, since we are dealing with relatively simple, grainy images, the features which the network needs to learn for successful classification are not especially complex in nature. In such a scenario, increasing the depth of the network to allow for the detection of more complex features may

not offer significant improvements in network performance. In future experiments, one may wish to check the relative performance of these networks on a data set with more varied, complicated images; for example, one could conduct a similar analysis on CIFAR-10 (Krizhevsky et al., 2009).

Another plausible alternative reason for the drop in performance is that using several layers of convolutions introduced precisely the form of translational invariance that we attempted to avoid by not using max-pooling. The size of the 2D cross section of the primary capsule layer is smaller for both of the convolutional architectures than it is in the original architecture, and therefore it is reasonable to assume that more information about the relative location of features has been lost through progressive convolutions. This has the potential to be especially problematic on a data set such as tiny small NORB, where there is explicit variation in pose between images.

3.4. Capsule Networks with Skip Connections

Skip connections are connections between non-adjacent layers of a neural network. Typically, in a neural network which includes skip connections, layer n receives its inputs from both layer $n-1$ and $n-2$. Skip connections have been included in state-of-the-art neural networks for image recognition and have improved the performance of neural architectures (He et al., 2016).

There is no widely accepted theoretical understanding for why skip connections are so successful, but a recent paper published earlier this year suggested that the inclusion of skips reduces the likelihood of elimination or overlap singularities arising in a network (Emin Orhan & Pitkow, 2018). These singularities arise when the output weights emanating from a node become set to zero (elimination singularity) or when two different nodes have the same set of input weights across the same receptive field (overlap singularity) (Emin Orhan & Pitkow, 2018). Both types of singularity adversely affect the performance of neural networks. It should be noted that this is likely to be more of a problem in deeper architectures (Emin Orhan & Pitkow, 2018), so it is unclear whether skip connections will have a similar positive effect on the relatively shallow capsule networks considered here.

We tested one model on the tiny small NORB data set with a modified skip connection skipping the primary capsule. To adjust the size, the skip also applied a regular convolution with kernel size 9×9, stride 2, 256 filters, and leaky ReLU. The result was added element-wise to the results of the primary capsule layer for use as input for the DigitCaps layer. A learning rate of 0.0001 was needed for the model to consistently learn. The resulting validation accuracy was $\mathbf{89.4 \pm 0.3}\%$.

This represents a minor improvement over the original architecture and was in fact the best performing architecture on tiny small NORB. Nonetheless, this was not a significant improvement. However, given our earlier comments about the applicability of skip connections to very deep architectures, they could be useful in deeper structures with multiple successive capsule networks of the sort described in Hinton et al.'s more recent paper on capsule networks (Hinton et al., 2018).

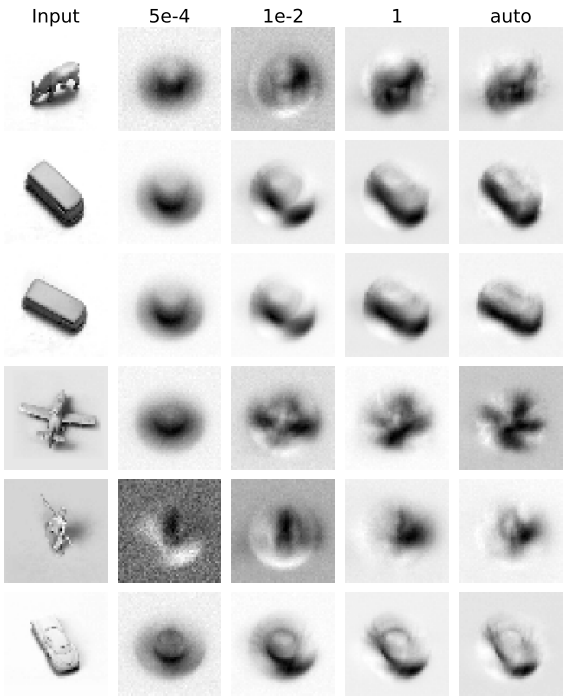

Figure 1. Effect of reconstruction error weight on CapsNet decoder output. Each row shows a different example from small NORB. The left-most column shows the original input images. The other columns show decoder outputs from models using different values of the reconstruction error α. The right-most column shows the output of an autoencoder CapsNet.

3.5. Reconstruction

Reconstruction is a core part of the CapsNets architecture, in part because the network has the capacity to summarise an input in a low dimensional vector, but also because CapsNets explicitly incorporate a reconstruction loss term as a part of the loss function that the network tries to minimise. Consequently, we investigated the reconstructions that CapsNets gave rise to.

Our CapsNet model routing layer outputs 5 length-16 vectors that each encode a particular class of object. The class associated with the vector with the highest norm is used as the prediction. In addition, the vector associated with that class is fed into a dense decoder that attempts to reconstruct the image, which is then used in the reconstruction component of the loss function.

Figure 2 shows an example of the routing output from one of our CapsNet models for several examples from small NORB. Looking at the rows, the first example shows the airplane vector is largest, though the truck, car, and animal

vectors also have a signal. This differs from the third example, where the truck vector has little competition. Another way to view this is column-wise. For the two human figures, their vectors differ but have similar shapes.

We can improve the quality of the reconstructed images by increasing the weighting α of the reconstruction error's contribution to the total loss. The baseline uses $\alpha = 0.0005$. The new model with $\alpha = 1$ had 92.6% accuracy on small NORB and the model with $\alpha = 0.01$ had 91.8% accuracy (though it hadn't reached early stopping due to time constraints).

An extreme form of this is to ignore the classification loss entirely and create a CapsNet autoencoder. Hinton mentioned using CapsNet autoencoders as a way of pretraining the capsules (Hinton, 2017). The decoded images from the length 16 vector corresponding to the correct class is shown shown in Figure 1.

We could compare this to a CNN autoencoder. We used a simliar architecture as our base CNN model from G63 (2018) but with a bottleneck layer with a length 16 vector, and dense layers of size 512, 1024, and 2304 like the CapsNet. In this case, the loss function is identical for both the CNN and the CapsNet. The CNN autoencoder reached 0.00335 loss. The CapsNet autoencoder reached 0.00309, and due to time constraints it had not early stopped.

While the CapsNet had gotten a lower loss than the CNN even before early stopping, the two might not be comparable. In the CapsNet, the decoder is given the class's vector by masking out the non-predicted classes. The input is of size 80, but all but 16 of those values are 0. This allows and requires the CapsNet's decoder's first layer to process each class differently. A more direct comparison would be to implement the same masking on the CNN autoencoder.

3.6. Multi-Task Learning

We also ran experiments using multi-task learning (MTL) on the small NORB data set with a CapsNet. In this case, along with predicting the class of the object, the model must also predict the azimuth of the object. Our idea was that this additional task would encourage the network to encode azimuth in its representation.

Azimuth is included in the original small NORB data set (LeCun & Huang, 2004). One issue with the azimuth data is that it is not aligned and two objects with the same azimuth could be facing in different directions. We adjusted this in our MTL version of the data set.

To adapt the Small NORB model from G63 (2018) to multi-task learning, we added a component to the loss based on the azimuth. In addition to the class, the model learns how to output a number $\theta_y \in [0, 1]$ where $360 \cdot \theta_y$ should predict the target angle $\theta_t \in [0, 360)$. It does this through a second decoder based on the routing output layer, with a dense layer of 512 units and leaky ReLU, followed by a dense layer with a single unit using sigmoid. The result is two dense decoders generating the decoded image and angle from the routing output layer.

The loss function becomes

$$L = \text{margin loss} + \alpha \cdot \text{reconstruction loss} + \beta \cdot \text{azimuth loss}, \tag{5}$$

where the margin loss and reconstruction loss are from the original CapsNet loss function described in Sabour et al. (2017).

We tested using the following azimuth loss

$$\text{azimuth loss} = \sin^2\left(\frac{\theta_y - \theta_t}{2}\right). \tag{6}$$

The performance on the original task was not improved and was a little degraded. Adjusting the loss component weights, we reached an accuracy of **92.9**% on Small NORB, while our baseline had reached 93.3%.

We also ran some qualitative experiments on this model. By adding an offset to the length-16 routing output vector and then feeding the new vector back into the decoder, one can see how the adjusting the values routing output vector affects the decoded images and azimuth. In the case of our MTL model, we can output both the reconstructed image and the azimuth. Since the azimuth should be encoded in the length-16 vector, it could be possible to adjust the vector in such a way that it rotates the decoded image. We attempted this via a few methods, such as adjusting each node individually, adding random offsets and viewing the decoded image and azimuth, or interpolating between two known images which is described in Figure 4. Unfortunately none of the approaches listed here gave satisfactory results. For example, in Figure 4, the 4th image from the left is what noisy decoded outputs look like, and the images before and after don't shift to the expected intermediate poses.

One issue with this loss function is that some objects, such as a bus, can appear symmetric. This means we can unfairly penalise the model for making a mistake humans make. We could try another loss function that takes that into account.

4. Related Work

CapsNets are a brand new construct, having first been implemented only 4 months ago, so there are not many papers on the topic yet. Those that do exist focus mainly on applying CapsNets to problems that traditional convolutional networks struggle with. For example, convolutional networks tend to work best when trained on large amounts of data: MNIST and small NORB both have around 50,000 images, while imageNet, another popular data set of labelled images, has 1.2 million. However, in some settings (for example, in medical settings, where fMRI imaging is time-consuming and expensive), data is naturally hard to acquire. While this is not something that we studied in our report, certain authors have found that CapsNets are

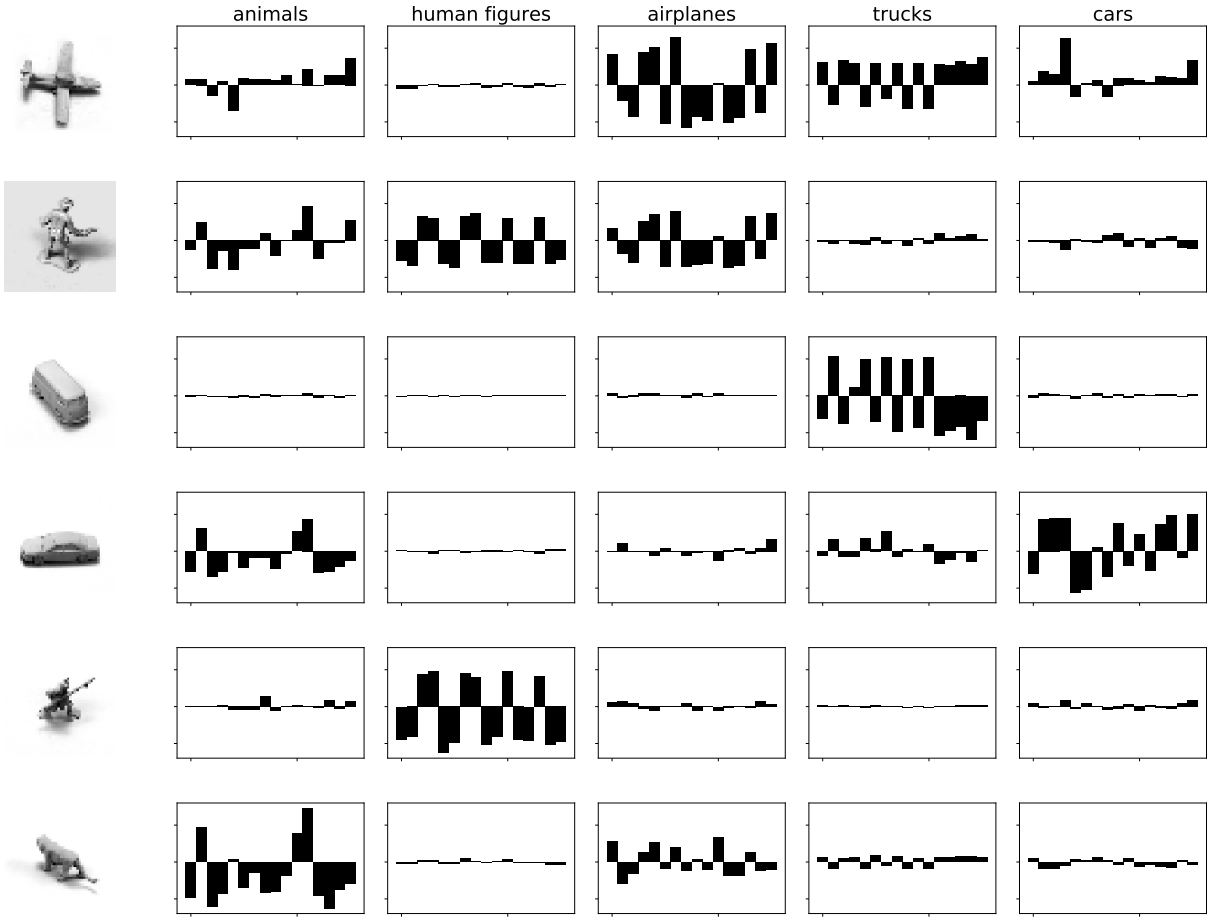

Figure 2. Capsule routing output for the six example images from the Small NORB validation set in the left column. Each row shows the example image, then bar plots of the values of the five length-16 vectors, with each subplot corresponding to a different class. At testing time, the prediction is chosen based on which of these vectors has the largest norm.

more effective in data-limited situations than traditional convolutional networks. For example, Afshar et al used a CapsNets architecture on a brain tumour classification task, rather than a convolutional network, because they only had around 3,000 labelled MRI brain scans, and thought that capsules were better suited for low-data tasks (Afshar et al., 2018). Similarly, Qiao et al used CapsNets on a stimulus recreation task, which involves reading fMRI brain scans of patients, and from that trying to decode what stimulus they were presented while being scanned. They only had around 3,000 labelled images, which is why they used Capsule networks instead of convolutional networks.

CapsNets have also been used in other situations. For example, Jaiswal et al implemented them as part of a generative adversarial network (GAN), by changing the discriminator network from a convolutional network to a CapsNet (Jaiswal et al., 2018). They found that this offered modest improvement over traditional GANs.

Furthermore, it is important to recognise the important role that neuroscience played in the development of CapsNets. In a lecture at MIT, Hinton explicitly stated that the structure of capsules was inspired by minicolumns in the brain (Hinton, 2017). The exact purpose of minicolumns is still debated in the neuroscientific community (Buxhoeveden & Casanova, 2002), but Hinton speculates that they perform the same sort of procedure that capsules do. He hypothesises that minicolumns implement some sort of routing operation between lower and higher level features: given a low-level feature, a minicolumn will activate another minicolumn responsible for the corresponding high-level feature. For example, the minicolumn responsible for detecting a facial feature (e.g. eyes) should redirect that input to the minicolumn responsible for detecting faces.

Moreover, the routing process involves low level capsules trying to predict the output of high level capsules. There is neuroscientific evidence that suggests that the brain operates in a similar fashion, with some neurons continuously trying to predict the way in which higher level neurons will fire, and adjusting their behaviour accordingly (Clark, 2013). Theoretically, one could argue that by constructing higher level capsules from a combination of lower level

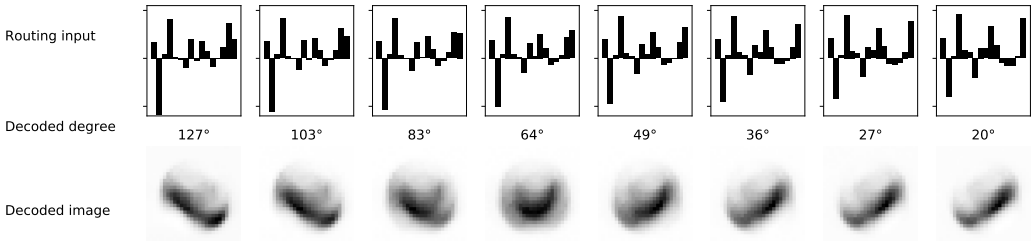

Figure 3. Interpolating between two routing outputs of a trained CapsNet on two of the inputs from Figure 4. The first and last column are the CapsNet's output on real inputs.

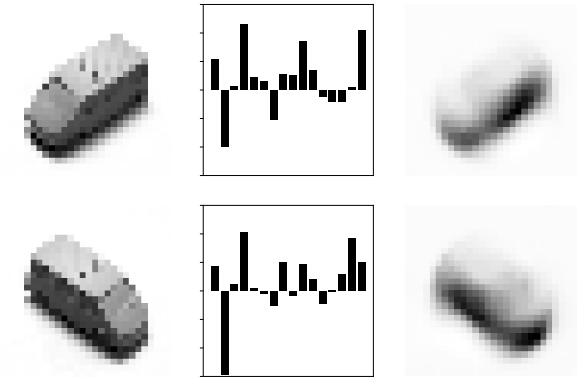

Figure 4. The three columns show two examples from small NORB of the same object with different azimuths, the values of the length-16 vector routing output for the truck class, and the reconstruction from our trained MTL model.

capsules, the network smooths over a certain amount of noise associated with each of the lower-level capsules; this is analogous to how the brain smooths out noise from noisy sensors (Adams et al., 2013).

Finally, there has been recent work by Hinton et al that tries to improve the routing algorithm from Sabour et al. (2017), by using expectation-maximisation routing (EM routing) (Hinton et al., 2018). The technical details for this routing technique are beyond the scope of this report. However, one particular aspect of the technique stands out: EM routing allows for the construction of networks with multiple consecutive convolutional capsule layers, making deeper architectures possible. This may allow them to better identify higher level features, and thereby increase performance. Deeper architectures have been effective in traditional convolutional networks, so it's reasonable to assume deep capsule networks may likewise work better than shallower alternatives (Wang et al., 2017).

5. Conclusions

Our initial experiments showed that modifying the length of the 'feature vectors' in the convolutional capsule layer does not make a significant difference to the network performance. However, in our implementation, networks with vectors of length 8 and 16 performed marginally better than networks with vectors of length 4. We attribute this to the fact that longer vectors are capable of encoding more complicated features. The addition of more convolutional layers at the start of the network also failed to improve performance.

Our most significant result was finding that the inclusion of a skip connection over the Primary Capsule layer led to a slight improvement in the average network performance. We stress that the improvement associated with skip connections may become more significant in deeper capsule architectures, such as those put forward in Hinton et al. (2018). This was our best result on the sub-sampled "tiny small NORB" data set, and our final classification accuracy was found to be 89.4%.

We also explored the ability of the CapsNet to encode information. We did this by qualitatively exploring the reconstructed images while adjusting the reconstruction loss weight. We also compared the loss of a CapsNet and CNN autoencoder, which could be an interesting way to compare the representational ability of the CapsNet's vectors.

We also tried a novel multi-task learning assignment that involves predicting the azimuth of small NORB images in addition to the margin and reconstruction loss. While this did not improve the accuracy, it provided an interesting way to interpret the representation in the routing output layer. Further work could focus on finding a method to continually rotate the decoded image by adjusting a set of the parameters in the routing output layer.

References

Adams, Rick A, Shipp, Stewart, and Friston, Karl J. Predictions not commands: active inference in the motor system. *Brain Structure and Function*, 218(3):611–643, 2013.

Afshar, Parnian, Mohammadi, Arash, and Plataniotis, Konstantinos N. Brain tumor type classification via capsule networks. *arXiv preprint arXiv:1802.10200*, 2018.

Buxhoeveden, Daniel P. and Casanova, Manuel F. The minicolumn hypothesis in neuroscience. *Brain*, 125(5): 935–951, 2002. doi: 10.1093/brain/awf110. URL http://dx.doi.org/10.1093/brain/awf110.

Clark, Andy. Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, 36(3):181–204, 2013.

Emin Orhan, A. and Pitkow, X. Skip connections eliminate singularities. In *International Conference on Learning Representations*, 2018.

G63, s1310324 s1779598 s1164250. Motivating capsule network architectures. *Unpublished, obviously*, 2018.

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

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. In *International Conference on Learning Representations*, 2018.

Hinton, Geoffrey. What is wrong with convolutional neural nets, 2017. URL https://www.youtube.com/watch?v=mpbWQbkl8_g#t=20m15s.

Jaiswal, Ayush, AbdAlmageed, Wael, and Natarajan, Premkumar. Capsulegan: Generative adversarial capsule network. *arXiv preprint arXiv:1802.06167*, 2018.

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

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

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.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted Boltzmann machines. In *Proc ICML*, pp. 807–814, 2010.

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

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Wang, Haohan, Raj, Bhiksha, and Xing, Eric P. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.