Research on Pruning Convolutional Neural Network, Autoencoder and Capsule Network

Tianyu Wang[1]

Australia National University, Colledge of Engineering and Computer Science
u6014854@anu.edu.au

Abstract. Some tasks, which are easy for a human, require complex neural network structures to solve. Although a large net structure usually leads to powerful model, it also has clear shortcomings. A large net structure requires more training time, suffers from overfitting more easily and is hard to deploy on mobile devices. The goal of network pruning is to find a balance between hardware requirement and task performance. Many works have been done trying to measure the neurons redundancy with cosine distance between weights. In this work, I applied pruning method with convolutional neural network (CNN), autoencoder and capsule network. The results show that the similarity measurement can be applied to CNN, autoencoder and capsule network.

Keywords: Network Pruning · Capsule Network · Convolutional Neural Network · Autoencoder

1 Introduction

1.1 Motivation

Nowadays, a neural network structure with a large number of parameters is not rare, especially in computer vision area where one single input can have thousands of features. Many successful applications show that large network structures usually yield better prediction performance.

The shortcomings of large network structures are also obvious. Large network structures directly lead to super high dimensional parameter space and a much longer training time is needed. Even after the model is trained, the prediction time will be long making it not suitable for real-time tasks. The training of a large network requires a large dataset. If the dataset is small or the structure of data is simple, large networks tend to overfit and are difficult to generalize to unseen data. Mobile devices usually have limited calculation power and storage space, making the deployment of large network structures difficult. Many strategies are proposed with the attempt to shrink the size of the network structure. Among those strategies, an effective one is to identify the redundancy in one network structure and prune useless neurons.

Intuitively, in a network with zero redundancy, neurons from the same layer must behave differently, in other words, detect and extract different features. If the weights of two neurons are similar to each other, then we may think their behaviours are roughly the same, and one of them is redundant. Thus, the similarity measurement can be regarded as an indirect redundancy measurement.

Weights similarity[2] is a successful similarity measurement. In the original paper, the weights similarity was only studied on a simple FNN with one hidden layer, and a positive correlation between the similarity and the number of hidden neurons was found. In this work, I extend the application of the pruning technique to CNN, CNN based autoencoders and recently published capsule network.

1.2 Background

CNN is a special neural network designed for 2D data and featured by convolution layer[4]. One convolution layer has multiple convolution kernels (filters), each kernel is a matrix. During convolution operation, each kernel scans through the whole input data with pre-defined step length and sum the results of element-wise multiplications at each step. Since convolution operation is differentiable, backpropagation algorithm applies. After proper training, each kernel captures a certain pattern, thus the similarity between kernels should also reflect the redundancy in a CNN.

Capsule network is another neural network designed for 2D data and featured by capsule layer[6]. There are two different capsule layers, primary capsule layer and digits capsule layer.

Primary capsule layer takes the input data and applies convolution operation. Denote the output of a convolution layer as Θ with the size $[W, H, C]$. In traditional CNN we interpret Θ as C feature maps, the resolution of each feature map is $[W, H]$. In capsule network we interpret Θ as a matrix of size $[W, H, N, V]$ where $C = V \times N$. That is, there are N capsule maps and in total $W \times H$ capsules in each capsule map, the dimension of each capsule is V, i.e., there are V elements in each capsule. A squash function squash the capsule norm into $(0, 1)$ before output.

The input of a digits capsule layer are capsules from last capsule layer. Routing by agreements algorithm[6] (detailed in the appendix) is applied to the input capsules and produce the output capsules. Every operation in the

routing by agreements algorithm is differentiable. Thus the backpropagation algorithm still applies. The output of a digits capsule layer are digits capsules. The output layer in a capsule network is usually a digits capsule layer with the number of output capsules equals the number of classes.

For a classification problem, a squash function squashes the norm of the capsules into range $(0, 1)$ before output and we treat the squashed norm as the unnormalized probability assigned to each class. Notice that the probabilities are not normalized, that is, the sum of the probabilities assigned to each class is not guaranteed to be one. What's more, the final output capsules contain high-level descriptions of the input data. That is to say we can reconstruct the input data using the digits capsules.

1.3 Dataset

All the experiments are done on Devanagari Handwritten Character Dataset (DHCD)[1]. The dataset contains 92,000 handwritten characters, each of them is a grey scale image with the resolution of 32 by 32. There are 46 classes and 2,000 images for each class. 85 percent images are randomly chosen for training, and the rest 15 percent is used for testing.

There are multiple reasons that this dataset is suitable for investigating pruning technique. The most important reason is that the quality of the data. As described above, data is evenly distributed among all 46 classes, which means the trained model is not likely to develop a bias towards specific class or classes. 2D data allows experiments with CNN model, CNN based model and capsule network and the weights can be visualized in an interpretable way. CNN based models naturally require more data to train convolutional kernels. A dataset of 92,000 images is a good start. Also, DHCD is a relatively simple dataset, a carefully designed CNN model can achieve an accuracy above 98 percent[5]. This difficulty is suitable for our research purpose.

1.4 Experiments Outline

The task the neural network models need to perform is classification task (classification is also the final goal for autoencoder). Handwritten classification is a classic application scenario for neural networks, and many different techniques are used to address this problem.

In this work, different neural network models are trained on DHCD. Initially, the models are given a relatively large number of neurons or kernels. During the experiments, neurons or kernels are gradually pruned, and the resulting smaller models are retrained. In this work, a wide range of pruning results are tested, and the corresponding performance is recorded and analysed.

The performance assessment criteria include classification accuracy, number of parameters, minimal cosine distance of weights. Then I compare my best model with a published result [5]. The results show that weights similarity measurement can guide the pruning on CNN, autoencoder and capsule network but the relation between the number of kernels (filters) and the redundancy is more complicated than negative correlation. Through pruning, near state-of-art classification performance is achieved with fewer parameters and low redundancy.

2 Methodology

In this section, the detail of similarity measurements and network structures design are introduced.

2.1 Weights Similarity

In a simple FNN, for neurons $n_{l,k}$ in layer l with weights $\mathbf{w}_{l,k,:}$, its output $\mathbf{a}_{l,k}$ is simply

$$\mathbf{a}_{l,k} = f(\mathbf{w}_{l,k,:} \cdot \mathbf{a}_{l-1,k})$$

where f is the activation function, $\cdot$ is vector inner product, $\mathbf{w}_{l,k,g}$ is the weight connecting $n_{l+1,k}$ and $n_{l,g}$, : means all indexes.

If two neurons $n_{l-1,i}$ and $n_{l-1,j}$ from layer $l-1$ with similar weights $\mathbf{w}_{l-1,i}$ and $\mathbf{w}_{l-1,j}$, i.e. $\mathbf{w}_{l-1,i} \approx \mathbf{w}_{l-1,j}$, then their output $\mathbf{a}_{l-1,i}$ and $\mathbf{a}_{l-1,j}$ should also be similar since the input $\mathbf{a}_{l-2,:}$ is the same. In this case, $\mathbf{a}_{l,k}$ can be calculated as

$$\begin{aligned}
\mathbf{a}_{l,k} &= f(\mathbf{w}_{l,:} \cdot \mathbf{a}_{l-1,:}) \\
&= f(\mathbf{w}_{l,:\neq ij} \cdot \mathbf{a}_{l-1,:\neq ij} + \mathbf{w}_{l,i} \cdot \mathbf{a}_{l-1,i} + \mathbf{w}_{l,j} \cdot \mathbf{a}_{l-1,j}) \\
&\approx f(\mathbf{w}_{l,:\neq ij} \cdot \mathbf{a}_{l-1,:\neq ij} + (\mathbf{w}_{l,i} + \mathbf{w}_{l,j}) \cdot \mathbf{a}_{l-1,i})
\end{aligned}$$

where $:\neq ij$ means all indexes excluding i and j.

This equation shows that if we treat $\mathbf{w}_{l,i} + \mathbf{w}_{l,j}$ as a new weight and remove $n_{l-1,j}$, $\mathbf{a}_{l,k}$ will not change much and the performance of the whole network is likely to remain unchanged. Thus, $n_{l-1,j}$ can be treated as redundancy and the weights similarity can serve as redundancy measurement.

In this work, cosine distance is used as the similarity measurements and for two vector $\mathbf{a}$ and $\mathbf{b}$.

$$cdist(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

The advantages of cosine distance is that the length of vectors are not taken into consideration, i.e. $cdist(\mathbf{a}, \mathbf{b}) = cdist(c_1\mathbf{a}, c_2\mathbf{b})$. If the difference between two weights is just a constant multiplier, $\mathbf{w}_{l-1,i} \propto \mathbf{w}_{l-1,j}$, above equation still applies with

$$\mathbf{w} \cdot c\mathbf{a} = c\mathbf{w} \cdot \mathbf{a} = \hat{\mathbf{w}} \cdot \mathbf{a}$$

Weights similarity is applied to all the networks tested in this work. For CNN, autoencoder and capsule network, the convolution kernels are flattened into vectors first, then the cosine distance between kernels are computed.

2.2 Data preprocessing

To prevent overfitting, data augmentation technique is used. Before being put into networks, images are cropped from 32*32 pixels into 28*28 pixels. The cropping centre is randomly chosen given that the four corners of the new images are inside of the old images. Under this constraint, for each image, there are $(32 - 28)^2 = 16$ unique and available cropping centres. Thus, theoretically, the dataset can provide $92,000 \times 16$ images.

To make gradients flow better the pixel values originally ranging from 0 to 1 are further normalized to -1 to 1 by

$$\hat{v} = \frac{v - mean}{std} = \frac{v - 0.5}{0.5}$$

Any normalized image can be restored by

$$v = \frac{\hat{v} + 1}{2}$$

3 Network Structure Design

3.1 Convolutional Neural Network

The structure of CNN model used in this work is specified as follows:

The CNN model contains three convolution layers, each of them are followed by max pooling, activation function and dropout.

As for activation function, Rectified Linear Unit (ReLU) is used to speed up the training process[8].

$$f(x) = max(x, 0)$$

The inputs are zero padded to 28+3 to preserve the image size and put into the first convolution layer. The kernel size of the first convolution layer is 7. It is followed by a 2*2 max pooling operation with stride 2, then a ReLU activation followed by a dropout layer with dropping rate 0.25. The output is sent into the second convolution layer.

The second convolution layer uses the kernel of size 5. The inputs are zero padded to 14+2. It is followed by another 2*2 max pooling operation with stride 2, then a ReLU activation and a dropout layer with dropping rate 0.25. The output is sent into the third convolution layer.

The third convolution layer uses the kernel of size 3. The inputs are zero padded to 7+1. It is followed by another 2*2 max pooling operation with stride 2, then a ReLU activation and a dropout layer with dropping rate 0.25. The output is sent to a fully connected layer which is the final layer.

The number of output channels of the first convolution layer is fixed to 6 and the number of output channels of the second and third convolution layers is initially set to 16 separately. Pruning is only applied to the number of output channels of the second and third convolution layers.

The highest kernel weights similarities are calculated and recorded during the pruning experiments.

3.2 Autoencoder

In autoencoder section, two different autoencoders are tested.

Normal Autoencoder The encoder contains three convolution layers of kernel size 3. The first and the second one are followed by max pooling, ReLU and dropout. The third convolution layer is only followed by a ReLU layer, the output of which is the latent representations of the input images. The settings of max pooling and dropout layer are the same as CNN model.

The decoder part contains three deconvolution (transposed convolution) layers. The first and the second one are followed by max unpooling, ReLU and dropout. The output of the third deconvolution layer is the reconstructed image, thus no other layer follows.

The whole autoencoder is symmetrically designed. The number of input and output channels of the first convolution layer is the same as the number of output and input channels of the third deconvolution layer. This is also the same for the second convolution layer and second deconvolution layer, the third convolution layer and the first deconvolution layer.

The number of output channels of the third convolution layer is fixed to 6 and the number of output channels of the first and second convolution layers are initially set to 16. Pruning is also applied symmetrically to the number of output channels of the first and second convolution layers, i.e. the number of input channels of the second and third deconvolution layers.

After the autoencoder is trained, the decoder will be replaced by a fully connected layer and the fully connected layer will be trained on classification task while the weights of the encoder are fixed. The highest kernel weights similarities are calculated and recorded during the pruning experiments.

Shared Weight Autoencoder To further regulate the autoencoder, I adopted the shared weight technique[7]. If we treat decoding as a strict inverse operation of encoding, then it makes sense to force the convolution layer, and the corresponding deconvolution layer share the same weights and perform inversely.

In the shared weight autoencoder, the convolution layer and the corresponding symmetrical deconvolution layer share the same weights. The rest setting and experiments method are the same as normal autoencoder.

3.3 Capsule Network

In this work, I construct a capsule network with one primary capsule layer and one digits capsule layer.

The primary capsule layer contains two convolution layer. The first convolution layer has kernels of size 9×9 followed by a Relu activation. The stride of the convolution operation is 1. The second convolution layer also has kernels of size 9×9. The stride of the convolution operation is 2. Then the output of the second convolution layer, i.e., a matrix of shape $[6, 6, C]$, is reshaped into size $[6 \times 6 \times N, 8]$ where C is the number of kernels and $C = N \times 8$ (we ignore the batch dimension for now). Now we get in total $6 \times 6 \times N$ capsules of dimension 8, and we squash them using the squash function introduced below. All the capsules from the primary capsule layer are sent into digits capsule layer. Through the routing by agreements algorithm, the digits capsule outputs 46 capsules of dimension 16, one for each class.

No pooling operation, dropout or zero padding is used.

The squash function used to introduce non-linearity is

$$\mathbf{v} = \frac{||\mathbf{s}||^2}{1 + ||\mathbf{s}||^2} \frac{\mathbf{s}}{||\mathbf{s}||}$$

where $\mathbf{s}$ is the input vector (capsule) of the squash function and $\mathbf{v}$ is the output squashed vector (capsule).

Since capsule network is designed to support multiple class prediction, softmax function is not powerful enough. Margin loss is used instead as

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

where L_k is the loss of the k^{th} class, $T_k = 1$ iff an object of class k is present in the input data, $\mathbf{v}_k$ is one of the output capsules of the digits capsule layer. The m^+ and m^- are the expected margin. We want the predicted probability of an existing object to be larger than m^+ and the predicted probability of a non-existing object to be less than m^-. As you can see, for an existing object once the predicted probability is higher than the m^+, the loss is zero. That is to say, the margin loss does not care about how high the probability is if the probability is higher than the margin. The λ decides the trade-off between assigning high probabilities to existing objects and assigning low probabilities to non-existing objects. In this work, I follow the original paper[6] and set $\lambda = 0.5$, $m^+ = 0.9$ and $m^- = 0.1$.

The number of kernels of the two convolution layers is initially set to 256. The highest kernel weights similarities are calculated and recorded during the pruning experiments.

4 Results and Discussion

Below are the results and observation of the experiments. In all experiments, batch size is set to 256, and the optimizer is adam[3] with weight decay rate equals 1e-6 and learning rate 5e-4. To display intuitively, all the results are shown in 3d coordinates. To reduce the influence of the noise, a Gaussian filter with $\sigma = 0.5$ is applied to smooth the data.

4.1 CNN test results and analysis

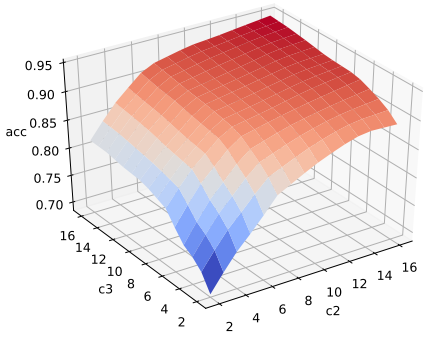

Fig. 1: CNN testing accuracy

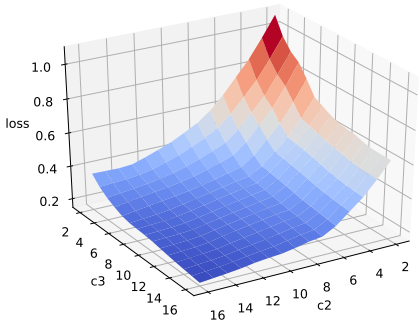

Fig. 2: CNN testing loss

For CNN, the pruning step length is set to one, i.e. one output channel or one kernel from one convolution layer is pruned at a time. The second and third convolution layers are initially set to 16 output channels (kernels) while the first convolution layer is fixed with 6 channels. The pruning in one convolution layer stops when there are only two output channels or two kernels left. In total 225 experiments are done.

With such large experiments scale (255 experiments in total), fine-tuning after pruning is not feasible. After each pruning operation, the network is re-initialized and trained. Early stopping is applied to prevent severe overfitting. The training is stopped when the testing accuracy reaches its peak.

CNN shows strong robustness. From Fig 1 and Fig 2 we can see that when $c2 \geq 7$ and $c3 \geq 6$, i.e., with more than 4056 parameters, the performance of CNN will only drop slightly after pruning. The c2 and c3 are the numbers of output channels in the second and third convolution layer. The number of parameters of a kernel is calculated as input_channels $\times$ output_channels $\times$ the kernel size. In this case the CNN has $1 \times c1 \times 7 \times 7 + c1 \times c2 \times 5 \times 5 + c2 \times c3 \times 3 \times 3 + 9 \times c3 \times 46$ parameters. The best accuracy is 0.94.

Fig 3 captures the changes of kernel weights similarity of the first convolution layer when the c2 and c3 are changed. Surprisingly, no simple positive or negative correlation is found. I speculate that the layers next to each other can compensate each other's work. If we carefully compare Fig 3 and Fig 4 (please notice that the angle of view of two figures are different, align them according to the axis, Fig 4 need to be rotated 180 degrees anticlockwise), we can see when the similarity of convolution layer one is relatively low, the similarity of convolution layer two is relatively high.

For Fig 4, except for its compensating relation with Fig 3, it is clear that lower c2 leads to higher minimum cosine distance of convolution layer two. The same pattern can be found on Fig 5 where lower c3 leads to higher minimum cosine distance of convolution layer three. However, the relation between c2 and the minimum cosine distance of convolution layer three is not trivial. When c3 is low, c2 is negatively correlated with the minimum cosine distance of convolution layer three, and when c3 is high, the correlation becomes positive. This phenomenon may be related to complex inter-influence between layers.

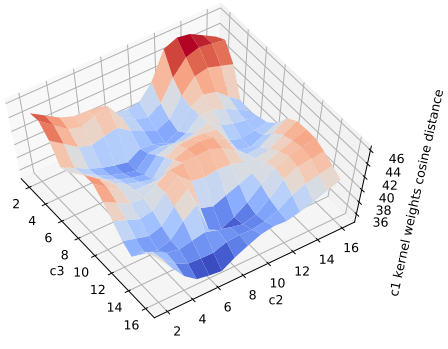

Fig. 3: CNN kernel weights cosine distance of the first convolution layer

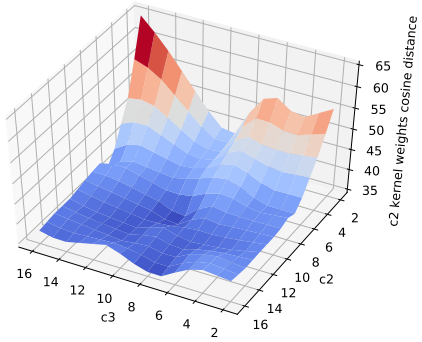

Fig. 4: CNN kernel weights cosine distance of the second convolution layer

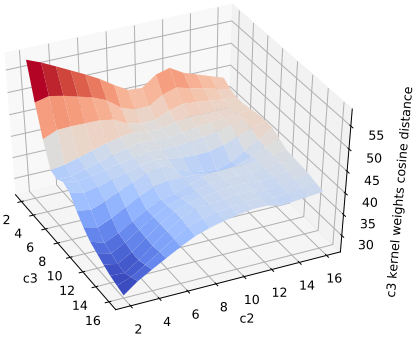

Fig. 5: CNN kernel weights cosine distance of the third convolution layer

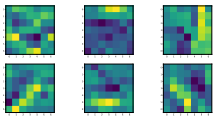

Fig. 6: convolution layer one weights with high cosine distance

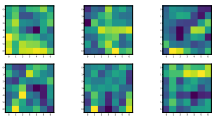

Fig. 7: convolution layer one weights with low cosine distance

Fig 6 and Fig 7 show the relation between the kernel cosine distance and the feature the kernels capture. While kernels in Fig 6 can be interpreted as finding the edge of different directions, kernels in Fig 7 is not really interpretable.

4.2 autoencoder test results and analysis

For autoencoder, the pruning step length is also set to one. The first and second convolution layers are initially set to 16 output channels while the third convolution layer is fixed with 6 channels. The pruning in one convolution layer stops when there are only two output channels left. In total 225 experiments are done.

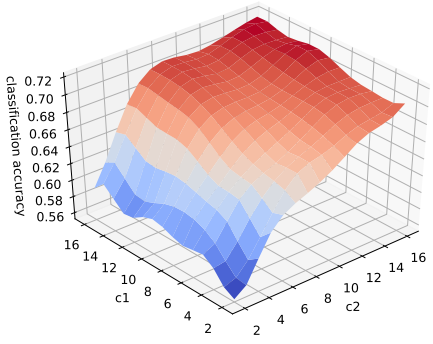

Fig. 8: Autoencoder testing accuracy

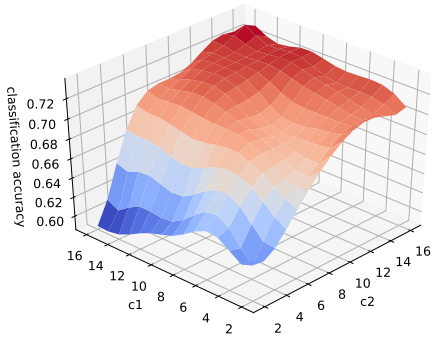

Fig. 9: Autoencoder with shared weights testing accuracy

Fig 8 and Fig 9 shows the comparison of classification accuracy between normal autoencoder and shared weights autoencoder. It is surprising that the shared weights autoencoder with only half number of weights achieved a performance similar to normal autoencoder. The robust range of autoencoder is $c1 \geq 9$ and $c2 \geq 13$, i.e., with more than 3672 parameters.

The best accuracy is 0.72. The performance of the autoencoder is largely out performed by the CNN model. It is possible that the features needed to accurately reconstruct the data are different to the features needed to accurately classify the data. Thus, if the data in dataset is evenly distributed, direct training may lead to a better result.

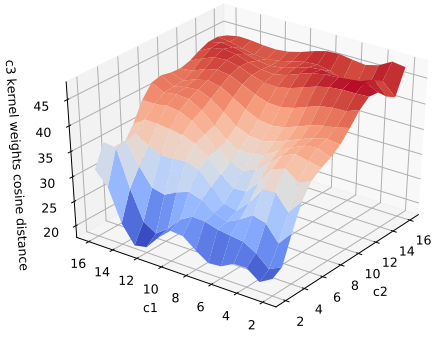

Fig. 10: cosine distance of the third convolution layer in normal autoencoder

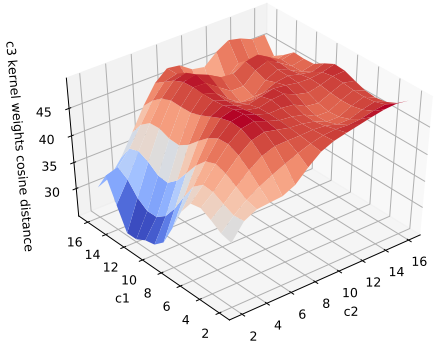

Fig. 11: cosine distance of the third convolution layer in shared weights autoencoder

Fig 10 and Fig 11 show the minimum cosine distance of the layers that directly generate the latent representation. Compared with Fig 8 and Fig 9, it is clear that high classification accuracy corresponds to a high minimum cosine distance. If we compare the minimum cosine distance between the normal autoencoder and the shared weights autoencoder, the shared weights autoencoder always has a larger cosine distance than the normal autoencoder.

4.3 Capsule network test results and analysis

For capsule network, due to the limit of my computational power, the pruning step length is set to 32, i.e. 32 kernels from one hidden layer are pruned at a time. Initially, the number of kernels in both convolution layer is set to 256. The pruning in convolution layers stops when there are only 32 kernels left.

Fig 12 and Fig 13 show the testing accuracy and testing loss throughout the pruning. The c1 and c2 are the numbers of kernels in the first and second convolution layers. Unlike previous results, there is no sign of any robust area. When c1 and c2 are both 256, the total number of parameters is $256*9*9+256*9*9+46*16*8*1152 = 6824448$,

which is Surprisingly large. However, the accuracy of this large model is also surprisingly good. The highest testing accuracy is 0.992.

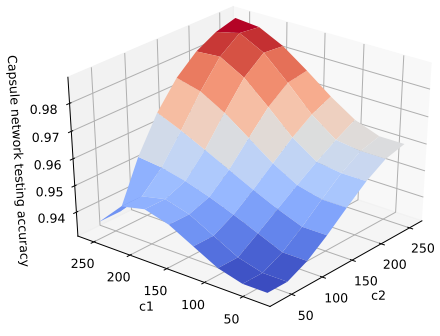

Fig. 12: Capsule network testing accuracy

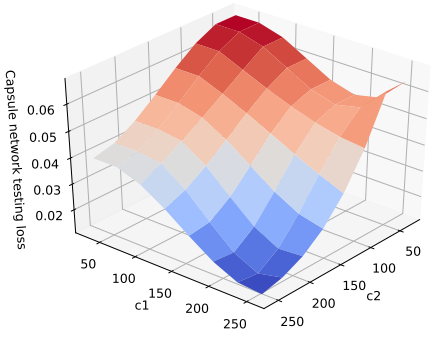

Fig. 13: Capsule network testing loss

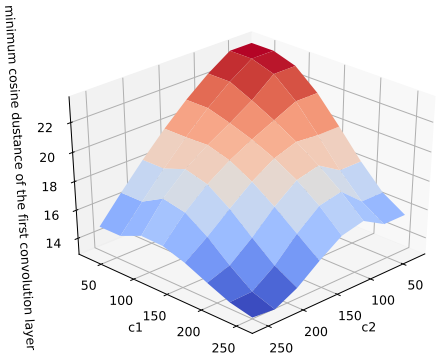

Fig. 14: Capsule network minimum cosine distance of the first convolution layer

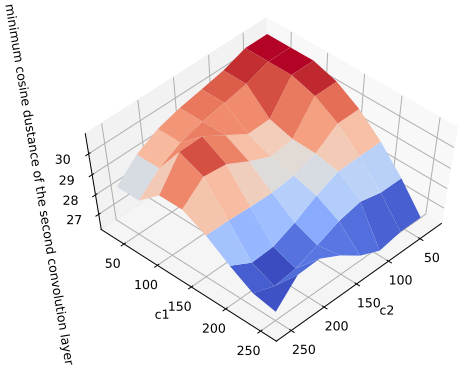

Fig. 15: Capsule network minimum cosine distance of the second convolution layer

Fig 14 and Fig 15 show the changes of the minimum cosine distance of weights throughout the experiments. As shown in Fig 14 and Fig 15, a negative correlation between c1 and the minimum cosine distance of the kernels of the first convolution layer is detected. It is reasonable since fewer kernels usually mean that each kernel needs to capture different features to maintain a comprehensive feature detection. Therefore, their behaviours should diverge. Also, a negative correlation is found between c2 and the minimum cosine distance of the kernels of the first convolution layer. The explanation could be that when the capacity of the second convolution layer is large, the features extraction task is mainly done by the second convolution layer. Thus the first convolution layer is less active.

According to Fig 15, with the same c1, lower c2 leads to a higher minimum weights cosine distance of the second convolution layer, which is the same as the first convolution layer. With the same c2, higher c1 also leads to a lower minimum weights cosine distance of the second convolution layer. A possible explanation is that if the first convolution has larger feature detection capacity, more unique inputs are extracted by the first convolution layer and send into the second convolution layer. As a result, the second convolution does not have to diverge too much to learn important features.

I think these results show that the feature extraction hierarchy of a deep neural network model can be strongly influenced by not only the number of layers but also the capacity of each layer. If the low-level layers do not have enough capacity to extract enough low-level features, then the high-level layers can compensate by extracting low-level features on their own with the cost of performance.

4.4 Comparing with published results

Now, I compare the best model in this works with a published result[5]. In this paper, a CNN model contains two convolution layers and two fully connected layers are specified. The detailed structure from bottom to top is:

i convolution layer with kernel size 5, output channel 4 and zero padding 2,
ii 2*2 max pooling layer with stride 2,
iii convolution layer with kernel size 5, output channel 12 and no zero padding,
iv 2*2 max pooling layer with stride 2,
v fully connected layer with 128 neurons and relu activation,
vi dropout layer with dropping rate 0.5,
vii output layer with 46 neurons.

The total number of parameters of the baseline model is 47892.

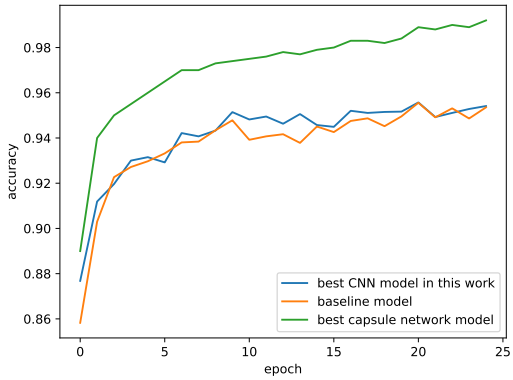

Fig. 16: performance comparison

The number of the parameters of the best CNN model in this work is 11622 and can be further reduced to 4056 with minor accuracy loss according to the robustness results. The number of the parameters of the best capsule network is 6824448. A large number of parameters result in a state-of-the-art classification accuracy on this dataset.

	accuracy	number of parameters
CNN baseline	0.95	47892
CNN ours	0.953	**11622**
Capsule network ours	**0.992**	6824448

Table 1: Testing results

5 Conclusion and Future Work

In this work, 829 experiments with different network structures are performed (255 on CNN, 510 on autoencoder, 64 on capsule network).

The best CNN model in this work successful reaches the same accuracy as the published model with 1/4 parameters and can be further reduced to 1/10 parameters with minor accuracy loss. The accuracy of the best capsule network achieved the state-of-the-art classification accuracy on the dataset. However the number of parameters is approximately 14 times larger than the baseline model and, as a result, the application scenario of capsule networks can be greatly limited. The autoencoder model cannot match the performance of the baseline model.

From results we can see, in a successful network structure, the weights similarity should fall into a certain range. A very low similarity (large cosine distance) shows that the model's capacity can not meet the needs. A very high similarity (small cosine distance) shows redundancy. However, the range is different from models to models and from layers to layers.

Results also show that the weight cosine distance can also effectively reflect redundancy in CNN models, autoencoders as well as capsule network, which means similarity measured by kernel cosine distance is suitable to guide

pruning on CNN kernels. In CNNs, convolution layers with meaningful and interpretable kernels tend to have large cosine distance. In autoencoder, the weights similarity of the last layer in an encoder directly reflects the quality of the latent representation.

Some interesting performance compensating behaviours between adjacent layers are revealed. This could mean that if the neural network's capacity is fixed, there is a trade-off between the structural complexity and the number of neurons.

In future, more thorough experiments should be done to study the interaction between layers and a deeper network should be used. If the relationship between the number of neurons and the structural complexity can be decided, we can prune more neurons and compensate it with a more complex structure or ignore design problems by stacking more neurons into the network.

6 Appendix

6.1 Routing by agreements algorithm

The inputs of the routing by agreement algorithm are capsules from previous layer. Then each input capsule $\mathbf{u}_j$ make a prediction $\hat{\mathbf{u}}_{j|i}$ on the output capsules $\mathbf{v}_j$ via a linear transformation $\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$. $\mathbf{W}_{ij}$ will be learned during the training.

Then the following routing procedure is applied.

Data: $\hat{\mathbf{u}}_{j|i}$, r number of iterations, layer l
Result: output capsule $\mathbf{v}_j$
for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$;
for *r iterations* **do**
  for all capsule i in layer l: $\mathbf{c}_i \leftarrow \text{Softmax}(\mathbf{b_i})$;
  for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \Sigma_i c_{ij}\hat{\mathbf{u}}_{j|i}$;
  for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \text{Squash}(\mathbf{s})_j$;
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$;
end

Algorithm 1: Routing procedure [6]

We can treat the routing procedure as an iterative coupling process. Each input capsule captures features according to its own receptive field. Then each capsule needs to be coupled with one output capsule. In the beginning, the unnormalized log prior coupling probability b_{ij} indicating that the input capsule $\mathbf{u}_j$ should couple with output capsule $\mathbf{v}_j$ is set to 0. In each iteration, a softmax function is used to calculate the normalized probability.

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

Then the output capsule $\mathbf{v}_j$ is calculated by combining the information from all its prediction $\mathbf{v}_j = \text{Squash}(\sum_i c_{ij}\hat{\mathbf{u}}_{j|i})$. The agreement between the output $\mathbf{v}_j$ and the predicted output $\hat{\mathbf{u}}_{j|i}$ is calculated by inner product $\mathbf{v}_j \cdot \hat{\mathbf{u}}_{j|i}$. If the agreement is strong then we have more evidence to think that the input capsule $\mathbf{u}_j$ should couple with output capsule $\mathbf{v}_j$. Thus we increase the unnormalized log probability by $b_{ij} = b_{ij} + \mathbf{v}_j \cdot \hat{\mathbf{u}}_{j|i}$.

In the iteration, coupling probability b_{ij} is updated according to $\mathbf{v}_j \cdot \hat{\mathbf{u}}_{j|i}$ then the output $\mathbf{v}_j$ is updated according to b_{ij}.

The number of iterations is a hyperparameter, and the typical choice is 3. After 3 iterations the final $\mathbf{v}_j$ is output.

References

1. Devanagari handwritten character dataset data set, 2016.
2. T. D. Gedeon. Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. In *Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 26–29, Nov 1995.
3. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
4. B. Yoshua LYann. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.
5. P. K. Gyawali S. Acharya, A. K. Pant. Deep learning based large scale handwritten devanagari character recognition. In *2015 9th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pages 1–6, Dec 2015.
6. Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.
7. J. Jin T. D. Gedeon, J.A. Catalan. Image compression using shared weights and bidirectional networks. 1997.
8. G E. Hinton V Nair. Rectified linear units improve restricted boltzmann machines. 2010.