Interpreting Capsule Networks for Image Classification by Routing Path Visualization

by
Aman Bhullar

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Mathematics and Statistics

Guelph, Ontario, Canada

ABSTRACT

INTERPRETING CAPSULE NETWORKS FOR IMAGE CLASSIFICATION BY ROUTING PATH VISUALIZATION

Aman Bhullar
University of Guelph, 2020

Advisor:
Dr. Ayesha Ali

Automating the classification of images is difficult in specialized areas because often there is insufficient labeled data to train a convolutional neural network, the most popular model for image classification [11]. Capsule networks, a neural network architecture proposed for image classification by Sabour et al. 2017, have been shown to require a smaller dataset than convolutional neural networks to train well [14]. In this thesis, a capsule network model that can classify astronomical images as containing or not containing at least one supernova light echo is identified, and shown to obtain an accuracy of 90% on the test set. In addition, *routing path visualization*, a technique for interpreting the entity that a given capsule in a capsule networks detects, is introduced in this thesis. Experimental results demonstrate that routing path visualization can also be used to precisely localize supernova light echoes in astronomical images.

Acknowledgements

I would like to express the deepest appreciation to my advisor, Dr. Ayesha Ali, for taking me on as a student, and believing in me throughout my degree. I would like to thank Dr. Doug Welch for providing the data for this thesis. And of course, I would like to thank my family and friends for their constant encouragement, support, and inspiration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rate at which data is generated has increased significantly in science, as well as virtually every other field. For example, by 2022, it is expected that 90 terabytes of astronomical data will be generated per night with the release of telescopes: Large Synoptic Survey Telescope, and Thirty Meter Telescope [8]. Accordingly, the development of models that are at least as accurate as humans but much faster at analysing data is crucial for rapid human advancement. Neural networks have become one of the most popular models because particular architectures have been found to achieve state-of-the-art (the best) performance on many problems. However, neural network models are often left as a black box which is particularly troubling when the application concerns human well-being.

In brief, this thesis contributes to the area of computer vision by introducing an interpretation technique called *routing path visualization* for capsule networks. Capsule networks are a newly proposed neural network for image classification that have been found to generalize well to novel viewpoints, and can be trained with a smaller training set than previous networks [14]. Routing path visualization identifies the regions of the input image that most influence a given capsule. In turn, the entity that each capsule detects can be interpreted, and it can be determined whether the correct information in the image is routed through the network. Thereafter, restrictions may be added to the network, or changes may be made to the training set to reduce the likelihood of failure.

Preventing failure is particularly important for applications where errors in prediction can lead to extremely unwanted events, such as in medical imaging or autonomous driving. For instance, the U.S. Food and Drug Administration wants all machine learning models for medical devices to be interpreted before they are put in production to ensure robustness [16]. Also, the electric car company, Tesla, trains their models on events that are anticipated to be problematic to reduce the likelihood of future model failure. Future problematic events are determined by model interpretation [3]. Moreover, Routing path visualization is shown to precisely localize the predicted class from an image when performed on a final layer capsule. In addition to being an interpretation technique, it can potentially be used for object localization.

Another goal of this thesis is to automate the detection of supernova light echoes in astronomical images. Light echoes are locations in the sky where interstellar dust has been illuminated by the brief outburst of a supernova. A capsule network model achieved 90% accuracy on the test set of astronomical images. Routing path visualization was then used to confirm that the final capsule network architecture used the correct regions of the image to make a prediction, and to localize light echoes in astronomical images.

Chapter 2 briefly reviews operations used in the relevant neural network architectures. Chapter 3 is a paper that introduces the routing path visualization technique and applies it to two real world datasets. Chapter 4 is a paper that presents the Python package, developed through this thesis for automating light echo detection. The capsule network model was trained on real world astronomical images to classify images as containing or not containing at least one supernova light echo. Finally, Chapter 5 provides conclusions and future work.

Chapter 2

Background

This chapter introduces convolutional neural networks (CNNs) and the processes implemented in the CNN and capsule network models presented in this thesis.

2.1 CNNs and Capsule Networks

Artificial neural networks were inspired by the biological neural system in that artificial neurons in a layer take as input the output of artificial neurons in the preceding layer, forming a network of artificial neurons. CNNs and capsule networks are popular neural network architectures for image classification. An example of a CNN architecture is given in Figure 2.1, and an example of a capsule network architecture is given in Figure 2.2. The total number of elements in the output vector of a CNN is equal to the number of classes for classification, and each element can be thought of as the probability that the image belongs to a particular class. The length of the output vector in a capsule network can be thought of as the probability that the image belongs to a particular class. Using the training set, gradient-based learning is used to find a set of weights that result in accurate classification.

The image is passed through a sequence of convolution layers in both CNNs and capsule

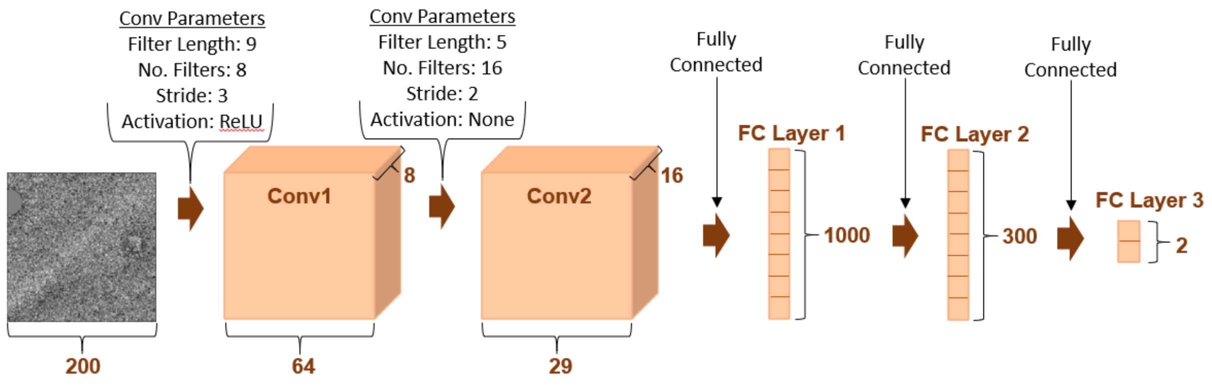


Figure 2.1: An example of a convolutional neural network architecture which contains 2 feature map tensors, labeled Conv1 and Conv2, and 3 fully connected (FC) layers. The final fully connected layer outputs a 2-dimensional vector.

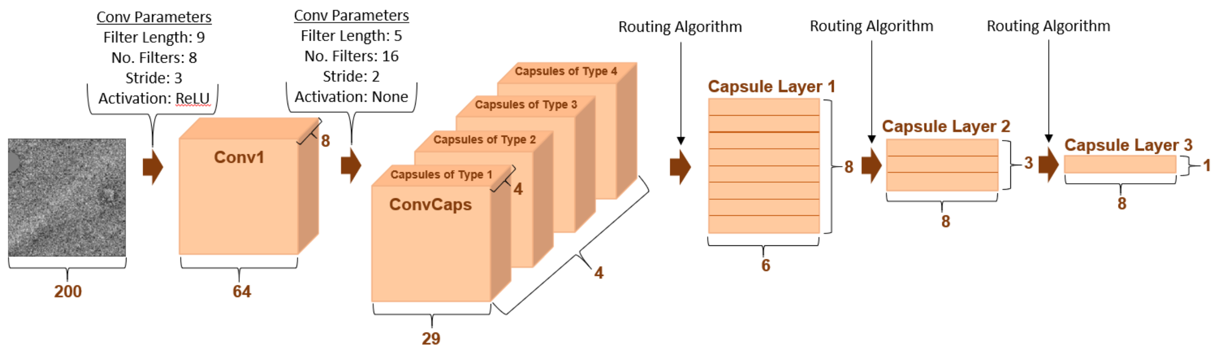


Figure 2.2: An example of a capsule network architecture which contains a feature map tensor, labeled Conv1, a ConvCaps layer, and 3 capsule layers. The final capsule layer outputs an 8-dimensional vector.

networks in order to obtain a tensor representation of the image which is better suited for classification ([11] and [14]). Each element of the tensor is called a neuron. The final tensor is referred to in CNNs as a feature map, and in capsule networks as a convolutional capsule (ConvCaps) layer. In CNNs, the tensor representation of the image is flattened to a vector of neurons and fed through at least one fully connected layer for classification [11]. In capsule networks, the tensor representation of the image is composed of vectors, called capsules, which are dynamically routed for classification to higher level capsules [14]. For different images, information of the input image is uniquely routed in capsule networks, whereas it follows a fixed path in CNNs.

2.2 Convolution Operation

The convolution operation takes as input a tensor of size $g_l \times g_w \times g_d$, a filter of size $f_l \times f_w \times g_d$, a stride parameter, and outputs a tensor of size $y_l \times y_w \times 1$. The output tensor expresses the similarity of the input to the filter. Typically, several filters of the same size are convovled with the input using the same stride, and the outputs of these convolutions are concatenated to a tensor of size $y_l \times y_w \times y_d$, where y_d is simply the total number of filters convolved with the input. This final output tensor is referred to as a feature map. The output length and width can be calculated using the following formula

$$y_i = \left(\frac{g_i - f_i}{s} + 1 \right) \tag{2.1}$$

where $i = l$ or w, and s is the stride [4].

An example of the convolution operation is shown in Figure 2.3. Starting from the top left-hand corner, the filter slides across and down the input tensor with a step size s, referred to as the stride. At each step, the overlapping elements of the filter and input are multiplied

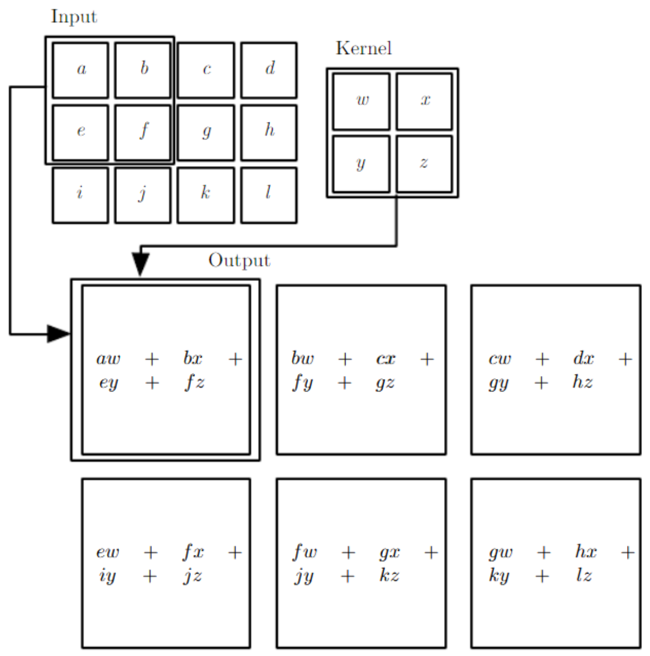


Figure 2.3: An example of the convolution operation on an input tensor of size $3 \times 4 \times 1$, a filter of size $2 \times 2 \times 1$, and a stride of 1. The filter is also referred to as a kernel. Illustration credit: page 330 of [4].

together, and then the products are summed to form the elements of the output tensor. The sum of products will be largest in the regions where the input is similar to the filter. A bias term, which is a real number, is added to all elements of the output tensor. Then a non-linearity function is applied to the output tensor to add complexity to the model. Two common non-linearity functions are ReLU, $F(x) = \max(0, x)$, and the sigmoid function, $S(x) = (1 + e^{-x})^{-1}$ [4]. The filter weights and the bias terms are learned during training.

2.3 Max Pooling

The max-pooling operation takes as input a tensor of size $g_l \times g_w \times 1$, a window of size $f_l \times f_w \times 1$, a stride parameter, and outputs a tensor of size $y_l \times y_w \times 1$. The output tensor represents the input with reduced quality. The length and width, y_l and y_w, of the output

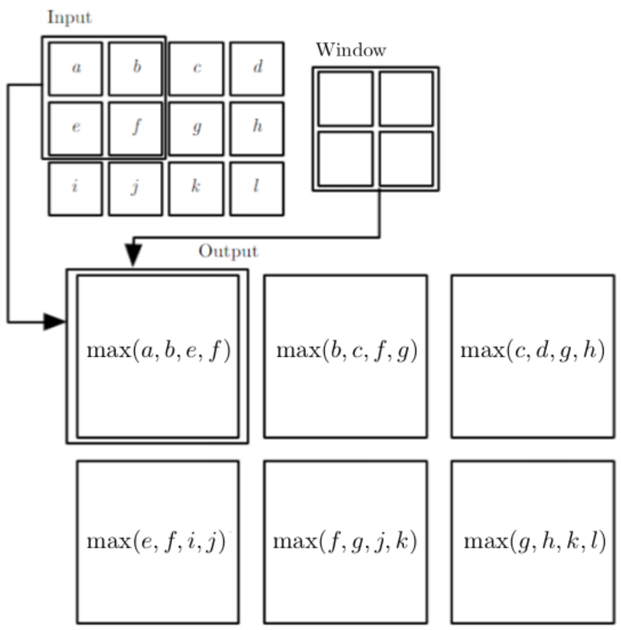


Figure 2.4: An example of the max-pooling operation on an input tensor of size $3 \times 4 \times 1$, a filter of size $2 \times 2 \times 1$, and a stride of 1. Illustration credit: page 330 of [4].

tensor are calculated using equation 2.1. An example of the max-pooling operation is shown in Figure 2.4. Starting from the top left-hand corner, the window slides across and down the input tensor with a step size s. At each step, the maximum element in the window is assigned to be an element of the output tensor. Max-pooling down-samples/compresses the input so that fewer parameters have to be learned during training. This reduces the size of the network which is ideal when optimizing for speed, however, it results in the loss of precise positional information. It is important to note that max-pooling does not contribute an additional set of weights to the network [4].

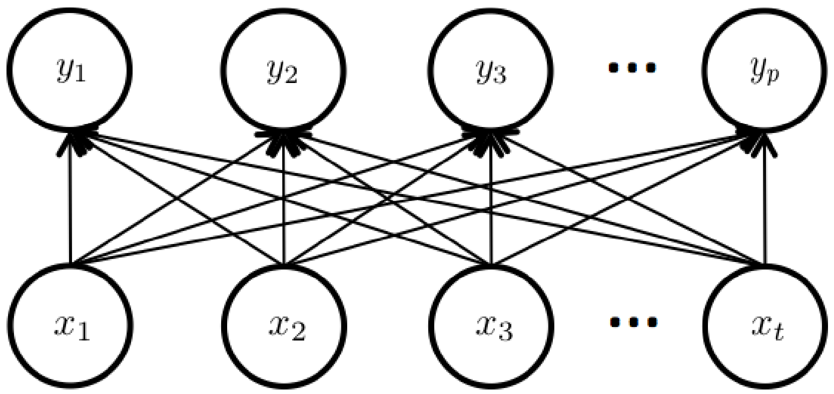


Figure 2.5: An example of a fully connected layer which requires t inputs and outputs a p dimensional vector.

2.4 Fully Connected Layer and Dropout

A fully connected layer takes a vector $\vec{x}$ as input, and returns a vector $\vec{y}$ as output whose dimension is not necessarily the same as the input. The output is calculated as

$$\vec{y} = f\Bigg(\mathbf{W}\vec{x} + \vec{c}\Bigg),$$

where $\mathbf{W}$ is the edge weight matrix, $\vec{c}$ is a vector of biases which are constants, and f is a non-linearity function. The edge weights and bias terms are learned during training. A visual example of a fully connected layer is shown in Figure 2.5. The edge weights are represented by the arrows, and the input and output elements, which are often referred to as neurons, are represented by the circles [4].

In CNNs, a fully connected layer follows the last feature map tensor. The tensor is flattened to a vector of real numbers, meaning it is collapsed into one dimension, and fully connected to an output vector. To increase the complexity of the model, a fully connected layer may follow a fully connected layer. For classification tasks, the last fully connected layer has as many output elements as classes that are to be predicted.

Fully connected layers significantly contribute to the total number of trainable weights in a neural network. Unfortunately, this results in an over-fitting issue when the training set is small. Dropout alleviates this issue by randomly omitting a certain percentage of fully connected neurons every forward propagation step whilst training. This makes it difficult for the network to simply memorize the training set, or learn unnecessary co-adaptations such as higher-level neurons correcting the mistakes of lower-level neurons ([6]).

2.5 Capsule Network Routing Algorithm

In capsule networks, the ConvCaps layer is the first layer of capsules, and these capsules are calculated by convolving the final feature map with a set of filters, as described in Section 2.2. Higher level capsules are calculated using the routing algorithm, whose pseudo-code is given below [14].

Algorithm 1 Routing Algorithm [14]

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $q$)
2:     for all capsule $i$ in layer $q$ and capsule $j$ in layer $(q+1)$: $b_{ij}^{(q)} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $q$: $\mathbf{c}_i^{(q)} \leftarrow \mathtt{softmax}(\mathbf{b}_i^{(q)})$.     ▷ softmax computes Eq. 2.2
5:         for all capsule $j$ in layer $(q+1)$: $\mathbf{s}_j^{(q)} \leftarrow \Sigma_i c_{ij}^{(q)} \hat{\mathbf{u}}_{j|i}^{(q)}$
6:         for all capsule $j$ in layer $(q+1)$: $\mathbf{v}_j^{(q)} \leftarrow \mathtt{squash}(\mathbf{s}_j^{(q)})$. ▷ squash computes Eq. 2.3
7:         for all capsule $i$ in layer $q$ and capsule $j$ in layer $(q+1)$: $b_{ij}^{(q)} \leftarrow b_{ij}^{(q)} + \hat{\mathbf{u}}_{j|i}^{(q)} \cdot \mathbf{v}_j^{(q)}$
8:         return $\mathbf{v}_j^{(q)}$
```

The extent to which each child capsule is coupled to each parent capsule is determined by the routing algorithm. The coupling coefficient between child capsule i in layer q and parent capsule j is denoted by $c_{ij}^{(q)}$, where $i = 1, 2, ..., I$ and $j = 1, 2, ..., J$. Figure 2.6 is a visual

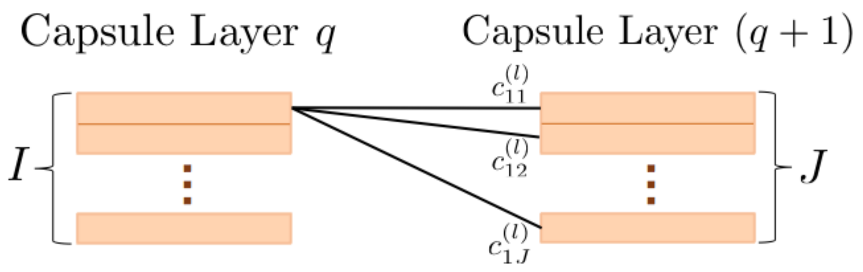


Figure 2.6: A visual representation of the coupling between capsule 1 in layer q and the capsules in the succeeding layer, where the lines connecting capsules symbolize the coupling coefficients.

representing the coupling between child and parent capsules. The coupling coefficients are calculated by

$$c_{ij}^{(q)} = \frac{\exp(b_{ij}^{(q)})}{\sum_j \exp(b_{ij}^{(q)})}, \tag{2.2}$$

where $b_{ij}^{(q)}$ are unnormalized log prior probabilities that capsule i should be routed to capsule j. Initially, a child capsule is evenly coupled to all parent capsules. After the routing algorithm is complete, a child capsule tends to send most of its output to parent capsules with other child capsules that have similar predictions. The prediction of child capsule i for parent capsule j is given by

$$\hat{\mathbf{u}}_{j|i}^{(q)} = \mathbf{W}_{ij}^{(q)} \mathbf{u}_i^{(q)},$$

where $\mathbf{W}_{ij}^{(q)}$ is a learned matrix that transforms $\mathbf{u}_i^{(q)}$, the vector output of child capsule i, to a prediction of parent capsule j. The input of parent capsule j is calculated by

$$\mathbf{s}_j^{(q)} = \sum_i c_{ij}^{(q)} \hat{\mathbf{u}}_{j|i}^{(q)}.$$

The vector output of parent capsule j is given by

$$\mathbf{v}_j^{(q)} = \frac{||\mathbf{s}_j^{(q)}||^2}{1 + ||\mathbf{s}_j^{(q)}||^2} \frac{\mathbf{s}_j^{(q)}}{||\mathbf{s}_j^{(q)}||}. \tag{2.3}$$

In essence, the routing algorithm calculates the output vector for each parent capsule by first finding how strongly each child capsule is routed to each parent capsule.

2.6 Gradient-Based Learning

Thousands of weights contributed by filters, edge weights, transformation matrices, and bias terms are learned such that they minimize a mathematical criteria called the loss function. For instance, margin loss is minimized to train capsule networks for image classification. Margin loss is given by

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

where $T = 1$ if the class is present in the image and 0 otherwise, $m^{+} = 0.9$, $||\mathbf{v}||$ is the length of the class-detecting capsule, $\lambda = 0.5$, and $m^{-} = 0.1$ [14]. Learning a set of weights that minimize the margin loss influences the length of the class-detecting capsule to be approximately 1 if the class is present in the image and 0 otherwise.

The vector of weights, $\mathbf{w}$, is updated in the direction of steepest descent of $L(\mathbf{w})$. This can be mathematically represented by

$$\mathbf{w}' = \mathbf{w} - \alpha\nabla_{\mathbf{w}}L(\mathbf{w}), \tag{2.4}$$

where $\nabla_{\mathbf{w}}L(\mathbf{w})$ is a vector such that each element k is the partial derivative of L with respect to w_k, and α, referred to as the learning rate, is the step size taken when updating $\mathbf{w}$ [4].

It is important to note that gradient-based learning methods often converge to a local minimum instead of the global minimum. In this work, the Adam optimizer, which is a gradient-based method of learning, was used to minimize the loss function [7]. The Adam optimizer has been demonstrated to achieve better performance than Equation 2.4 on problems with noisy gradients [7]. Training is terminated when the model begins over-fitting to the training data, as shown by a small training loss and large validation loss.

Chapter 3

Interpreting Capsule Networks for Image Classification by Routing Path Visualization

Amanjot Bhullar[*], R. Ayesha Ali[*], Douglas L. Welch[a]

[*]Department of Mathematics and Statistics University of Guelph, Canada

[a]Department of Physics and Astronomy McMaster University, Canada

3.1 Abstract

Artificial neural networks are popular for computer vision as they often give state-of-the-art performance, but are difficult to interpret because of their complexity. This black box modeling is especially troubling when the application concerns human well-being such as in medical imaging or autonomous driving. In this work, we propose a technique called routing path visualization for capsule networks, which reveals how much of each region in an image is routed to each capsule. In turn, this technique can be used to interpret the entity that

a given capsule detects. We demonstrate our new visualization technique on the MNIST dataset, and a set of astronomical images. Experimental results suggest that routing path visualization can be used to precisely localize the predicted class from an image. This is interesting because the capsule networks are trained using just images and their respective class labels, and without an additional set of information defining the location of the class in the image. Sample routing path visualization code is available for the MNIST dataset in Python from the GitHub repository
(https://github.com/AmanjotBhullar/Routing-Path-Visualization).

3.2 Introduction

Convolutional neural networks (CNNs) have achieved state-of-the-art results on many computer vision problems, and are currently the most popular algorithms for image classification. However, CNNs suffer from some limitations that are suspected to prevent them from being adopted in the long term. These limitations include the difficulty of CNNs to generalize to novel viewpoints and preserve the precise positional information of entities in an image [14]. The max-pooling operation, which is used in CNNs to achieve greater performance, is a source for the loss of precise positional information as it down-samples the image [4]. The human visual system does not suffer from these issues, and for that reason, CNNs may not be a great direction for computer vision. Deeper CNN architectures are often used to mitigate the impact of these limitations. Unfortunately, deeper networks contain more weights, and thus, require larger datasets to train. Automation can then become difficult in specialized fields.

Recently, Sabour et al. (2017) proposed capsule networks as an alternative neural network for computer vision, and achieved state-of-the-art image classification results on the MNIST dataset [10]. Capsule networks were created to overcome the limitations of CNNs, and are considered to more closely mimic the human visual system. They use a dynamic routing-by-agreement algorithm between layers which allows them to better model hierarchical relationships. In turn, this allows capsule networks to generalize better to novel viewpoints. Also, precise positional information is preserved as max-pooling is not implemented. Furthermore, capsule network architectures typically require fewer trainable weights than CNNs, and as a result, are expected to require a smaller set of images to train.

What the individual dimensions of a given capsule represent are interpreted in Sabour et al. (2017) and Shahroudnejad et al. (2018) [15]. In short, the image is reconstructed using the decoder network after some of the dimensions of the appropriate capsule(s) have been slightly perturbed [14]. What each dimension represents can be inferred from the way the perturbations affect the reconstructed image. However, asking what entity a given capsule detects is another interesting question. This was not a natural question for Sabour et al. (2017) and Shahroudnejad et al. (2018) because each MNIST image contains a solitary entity, a single digit. In real world applications this may not be the case. For example, astronomical images may contain several entities such as light echoes, saturated star images and optical path artifacts, within a single image. Interpreting the entities that capsules detect is useful for real world datasets as it can be determined if the correct information from the image is being used to predict the class of the image. Further, circumstances where the network may fail can be identified. Once identified, constraints can be applied to the network, or improvements can be made to the training set to decrease the probability of failure. Preventing failure is especially important for applications in which prediction errors can lead to extremely undesirable events, such as in medical imaging or autonomous driving.

The child-parent routing paths that are inherently created in capsule networks can be used to understand which areas of the input image most influence a capsule. In this work, we introduce a method called routing path visualization that reduces the child-parent routing paths to an image. This visualization shows the regions of the input image that influence a capsule of choice. Routing path visualization can be performed on the test set to interpret the entities that a capsule detects, thereby providing a visualization of a particular capsule for each image in the test set. Entities that consistently appear in the routing path visualization images of a particular capsule should be the entities that the capsule detects.

Routing path visualization does not require an additional network, such as a decoder network, to be attached to the capsule network, or increase the number of trainable weights of the capsule network. Instead it utilizes the child-parent routing paths that are inherently created in capsule networks. Routing path visualization generates a map which displays how strongly each region of an image is routed to a given capsule. Consecutively, it can be used to interpret the entity that a capsule detects. In contrast, routing path visualization does not interpret the information that each dimension of a capsule encodes, but instead reveals how much of each region in the image is routed to a capsule. To the best of our knowledge, there is not a technique comparable to routing path visualization.

Experimental results suggest that routing path visualization can be used to precisely localize the predicted class from an image when performed on a final layer capsule. This suggests that routing path visualization can also be used for object localization in addition to being an interpretation technique. Localization of the class is useful in medical imaging as the image expert can cross-check with the routing path visualization image to find abnormal areas. For instance, routing path visualization may find areas of the lung to correspond to lung disease that the image expert may miss. To the best of our knowledge, not only has the routing path that capsule networks inherently create not been used to interpret the entity that a capsule detects, but there is not a method of interpreting the entity that a capsule detects. Our method can be applied to all capsules in the network.

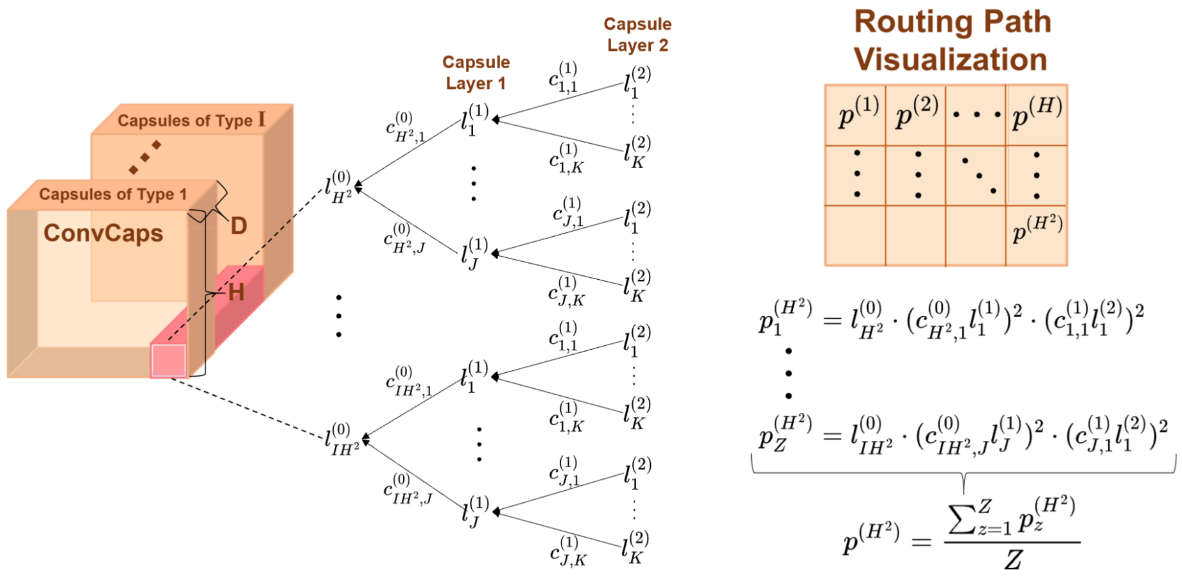
3.3 Background

In both CNNs and capsule networks, the image is passed through a series of convolutional layers to obtain a tensor representation of the image that is better suited for classification ([14], [11]). Each element of the tensor is called a neuron. The tensor is refered to as a feature map in CNNs and as a convolutional capsule (ConvCaps) layer in capsule networks.

In CNNs, the tensor representation of the image is flattened to a vector of neurons and fed through at least one fully connected layer for classification [11]. Every neuron in the tensor is connected to every neuron in the layer above. The edge weights connecting neurons in adjacent layers are learned during training, and do not change from image to image.

In capsule networks, the tensor representation of the image is composed of vectors, called capsules [14]. A capsule is a group of neurons that detects a specific entity in the image, and encodes the instantiation parameters of the entity if it is present. An entity is defined as an object or object part, and the instantiation parameters can include information on the position, size, or orientation of an entity. A capsule can have a Euclidean length of at most one, and is active if its length is close to one. The length of capsule i in layer q is denoted as $l_i^{(q)}$. A capsule's length can be viewed as the probability that the entity associated with that capsule is present. The tensor representation of the image is flattened to a vector of capsules which are routed to at least one layer of capsules for classification. Every capsule in the tensor is routed to every capsule in the layer above. The coupling coefficients determine the extent to which each child capsule is routed to each parent capsule. The coupling coefficient between child capsule i in layer q and parent capsule j is denoted by $c_{ij}^{(q)}$. (The coupling coefficients are not learned during training, and change from image to image.)

The coupling coefficients are calculated by a routing algorithm which takes as input the set of transformation matrices that are learned during training. Each child capsule in the tensor predicts the instantiation parameters of each parent capsule using the transformation



(a) Tree diagram of routing paths

(b) Path back-tracking for visualization

Figure 3.1: Routing path visualization for a ConvCaps layer with grid size $H \times H$ and I capsule types, and two succeeding capsule layers containing a J and K number of capsules, respectively. In this example, only the pixel in position H^2, $p^{(H^2)}$, of the visualization image is calculated. The image is used to precisely localize the object that capsule 1 in capsule layer 2 detects.

matrices. The prediction of child capsule i in layer q of parent capsule j is denoted by $\hat{\mathbf{u}}_{j|i}^{(q)}$. A child capsule tends to send most of its output to parent capsules for which the child's prediction is similar to the predictions of the other child capsules. When the predictions of multiple child capsules agree, a parent capsule becomes active, depending on the architecture of the capsule network. The input of a parent capsule is given by

$$\mathbf{s}_j^{(q+1)} = \sum_i c_{ij}^{(q)} \hat{\mathbf{u}}_{j|i}^{(q)}, \qquad i = 1, ..., IH^2 \quad \text{and} \quad j = 1, ..., J. \tag{3.1}$$

Figure 3.1 (a) reveals a diagram of the routing paths taken by capsules in a network that contains one ConvCaps layer and two capsule layers.

3.4 Methods

3.4.1 Weight Sharing

The feature map is convolved with a set of $N{=}I{\cdot}D$ filters to produce a ConvCaps layer of size $H{\times}H{\times}ID$, which contains $H{\cdot}H{\cdot}I$ capsules of dimension D. Each filter that was used to create the ConvCaps layer detects a particular feature in the feature map. Each capsule in an $H{\times}H$ grid was created using the same subset of D filters, and as a result, the capsules in a grid detect the same entity. The ConvCaps layer can be thought of as containing I capsule types, each of which detect a certain entity.

The convolution operation contains a form of weight sharing such that local features of the input are detected by the same filter(s). This weight sharing is natural because, as one could argue, the same criteria for detecting features in one location should be applied to other locations of the image [11]. The form of weight sharing in the convolution operation is analogous to the weight sharing we propose for routing in capsule networks.

It seems natural to subject capsules of the same type to the same criteria when being routed to capsules in the layer above. Hence, capsules of the same type should use the same transformation matrices for predicting entities in the layer above. This weight sharing not only leads to a simpler model, with fewer trainable weights, but also results in a more flexible model in which the number of weights no longer depends on the size of the input image. Consequently, the capsule network can handle images of any size. For instance, consider ConvCaps layer A. Suppose the layer succeeding A contains J capsules. Without weight sharing, there is a total of $H{\times}H{\times}I{\times}J$ transformation matrices associated with ConvCaps layer A, a set of J transformation matrices per capsule in A. With weight sharing, there is a total of only $I{\times}J$ transformation matrices associated with ConvCaps layer A, a set of J transformation matrices per capsule type in A.

In Sabour et al. (2017), each capsule in the ConvCaps layer, also known as the primary capsule layer, was given a unique set of transformation matrices. In this work, capsules in the ConvCaps layer that are of the same type share a set of transformation matrices. This weight sharing technique was created independently from LaLonde and Bagci (2018) which seems to use a similar weight sharing technique [9].

3.4.2 Routing Path Visualization

When a well-trained capsule network predicts the class of an image, the capsules in the ConvCaps layer that contain information relevant to the prediction become active, and some capsules containing irrelevant information also become active. Active capsules containing irrelevant information should be routed to at least one inactive capsule in succeeding layers as inactive capsules weakly affect the final prediction. In more detail, the predictions, $\hat{\mathbf{u}}_{j|i}^{(q)}$, of inactive capsules have small lengths, and as a result, the individual elements of $\hat{\mathbf{u}}_{j|i}^{(q)}$ are small. Equation 3.1 suggests that the predictions of inactive capsules contribute very little to $\mathbf{s}_j^{(q+1)}$ as $c_{ij}^{(q)}\hat{\mathbf{u}}_{j|i}^{(q)}$ will be small if $||\hat{\mathbf{u}}_{j|i}^{(q)}||$ is small.

Moreover, the ConvCaps layer preserves the relative positions of objects in the image. For example, the information of an object in the upper right corner of an image will remain in the upper right corner in the ConvCaps layer. Hence, if the predicted class is the existence of an object then the ConvCaps capsules that are positioned relative to the object's position in the image should become active as they contain relevant information. Intuitively, in a well-trained network these relevant capsules are routed to the capsule in the final layer that predicts the existence of the object, and all other active capsules are routed to inactive capsules in succeeding layers. Ergo, the child-parent routing path can distinguish between active capsules containing relevant and irrelevant information. Consequently, the object can be precisely localized by filtering out irrelevant active capsules in the ConvCaps layer. This filtering out procedure is refered to as *routing path visualization*. In turn, the entities that capsules detect can be interpreted, and situations where the network may fail can be identified.

Figure 3.1 shows an example of routing path visualization for a ConvCaps layer with grid size $H \times H$ and I capsule types, and two succeeding capsule layers containing J and K capsules, respectively. In this example, the entity that is detected by capsule 1 in the last capsule layer is visualized. This is achieved by calculating each pixel, $p^{(h)}$, of the routing path visualization image and given by

$$p^{(h)} = \frac{\sum_{z=1}^{Z} p_z^{(h)}}{Z}, \tag{3.2}$$

where $h=1, ..., H^2$, and Z is the total number of ways the capsules in grid cell h of the ConvCaps layer can be routed to capsule 1 in the last capsule layer. Further,

$$p_z^{(h)} = l_i^{(0)} \cdot (c_{ij}^{(0)} l_j^{(1)})^2 \cdot (c_{j1}^{(1)} l_1^{(2)})^2. \tag{3.3}$$

Since the lengths of the capsules and the coupling coefficients are always less than 1, $p_z^{(h)}$ is always less than 1.

Equation 3.3 measures how much of capsule i in the ConvCaps layer is routed to capsule 1 in the second capsule layer through a particular path. The term $l_i^{(0)}$ will be large if capsule i in the ConvCaps layer is active, the term $(c_{ij}^{(0)} l_j^{(1)})^2$ will be large if capsule i in the ConvCaps layer is mostly routed to an active parent capsule j, and the term $(c_{j1}^{(1)} l_1^{(2)})^2$ will be large if capsule j in the first capsule layer is mostly routed to capsule 1 in the second capsule layer. Equation 3.2 normalizes this value over all paths of all capsules in grid cell h.

	Convolutional Layers		**Capsule Layers** (C, D)			
Data	**Feature Map** (F, N, S)	**ConvCaps** (F, I, D, S)	1	2	3	**Weights**
MNIST	(3, 256, 1)	(3, 32, 8, 1)	(10, 16)			633,600
CFHT	(9, 8, 3)	(5, 4, 4, 2)	(8, 6)	(3, 8)	(1, 8)	5,984

Table 3.1: A summary of the capsule network architecture that was trained on the MNIST and the CFHT dataset. Models are defined by the feature map, ConvCaps layer, and capsule layers, where C is the number of capsules in the layer, D is the number of dimensions per capsule, F is the length of each filter, N is the number of filters used, S is the stride, and I is the number of capsule types. The Weights column lists the total number of trainable weights in each model.

3.5 Experiments

We demonstrate routing path visualization on a capsule network that classifies the MNIST dataset [10], and on a capsule network that classifies astronomical images as containing or not containing at least one supernovae light echo [12].

The capsules in the final capsule layer are interpreted for the former capsule network, and to show that routing path visualization can also be used to interpret intermediate layer capsules, the capsules in capsule layer 2 are interpreted for the latter capsule network. The architecture for both models is given in Table 3.1. The models were trained using a single NVIDIA Pascal P100 GPU on TensorFlow [2] with the Adam optimizer [7]. For both models, margin loss was minimized during training without an additional reconstruction loss [14]. Although the resulting capsules in the final layer may not encode all instantiation parameters of the class, routing path visualization still works because it interprets the entity that a capsule itself detects, not what the individual dimensions of the capsule represent. It is important to mention that reconstruction loss, which measures the difference between the input image and the reconstructed image, acts as a regularization term and encourages the model to learn a more optimal set of weights [14]. For this reason, reconstruction loss should

		Accuracy		Training		
Data	**Weights**	**Val**	**Test**	**Duration**	**Steps**	**Time**
MNIST	633,600	92%	91%	12h	1600	0.11s
CFHT	5,984	88%	92%	< 1h	9200	0.015s

Table 3.2: Prediction accuracy for the validation and test sets. The time taken to train each model is given under the Duration column. The total number of times the weights were updated during training is given under the Steps column. The time taken to classify a single image is given under the Time column, and this number is averaged over 50 images.

be used for improving model accuracy, but not for capsule entity interpretation.

The summary statistics for each model are reported in Table 3.2. The accuracy was calculated using the following formula

$$\frac{\texttt{True Positives+True Negatives}}{\texttt{No. of Images in Set}} \times 100\%.$$

For the MNIST dataset, the image was classified to contain the digit whose associated capsule had the largest euclidean length. For the CFHT dataset, the image was classified as containing a light echo if the length of the light-echo-detecting capsule was greater than 0.5; this cutoff was found to work well in practice.

3.5.1 MNIST

The capsule network that was trained using the MNIST dataset has a similar architecture to the network in Sabour et al. (2017), and the architecture is given in Table 3.1. The main difference here is that the network uses filters of size 3 instead of 9 because the ConvCaps layer does not preserve the relative positions of objects when it has a grid size that is much smaller than the size of the image. Due to computational time, training was stopped when the validation accuracy was good as opposed to state-of-the-art as the objective was to evaluate routing path visualization, not prediction accuracy per se.

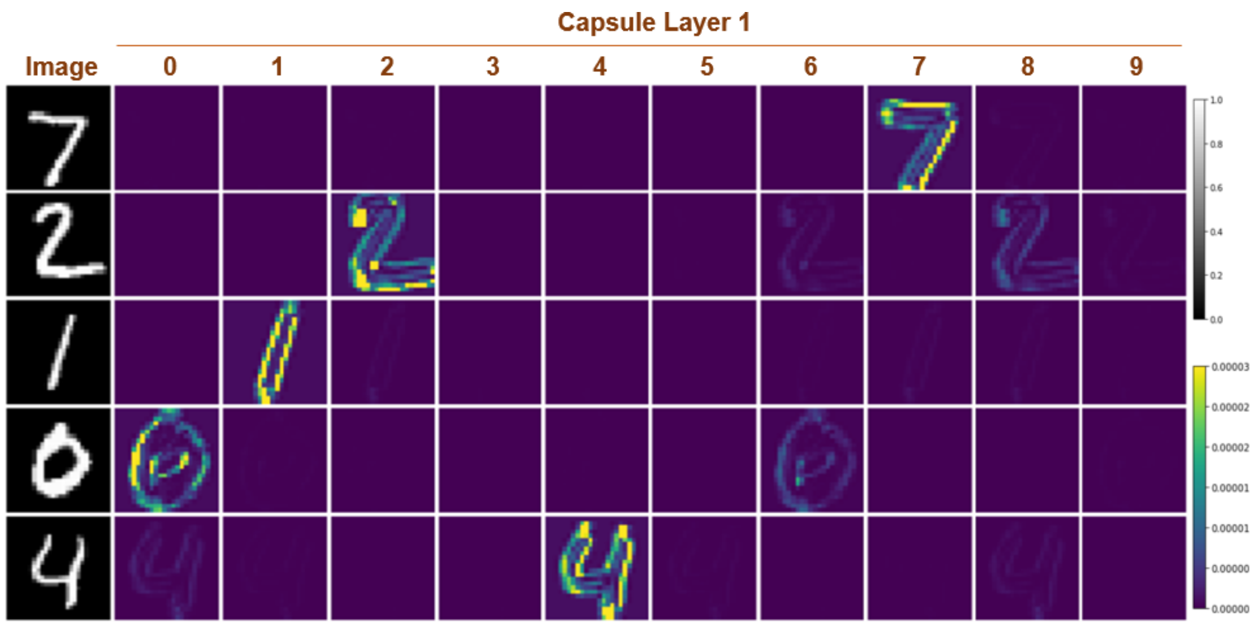


Figure 3.2: Routing path visualization to interpret the entities that the capsules in capsule layer 1 detect. Sample test images are of size 28×28, and routing path visualization images of capsule layer 1 are of size 24×24.

Routing path visualization was used on the test set to interpret the entities that are detected by the capsules in the capsule layer. The results from a sample set of 5 images is shown in Figure 3.2, and are representative of the results on all images in the test set. In general, the contour of the digit in an image is routed to the correct digit capsule. This is explainable as the contour of a digit provides the most useful information as all digits are the same colour. The regions of the image containing the 2 and 0 are routed to digit capsules 8 and 6, respectively (Figure 3.2). This finding suggests that the network may, at times, have difficulty distinguishing a 2 from an 8, or a 0 from a 6.

3.5.2 CFHT

An astronomical survey was performed by the Canada-France-Hawaii Telescope (CFHT) charge-coupled device mosaic imager, MegaCam, in 2011. The primary objective was to search for supernova light echoes [12]. Light echoes are locations on the sky where inter-

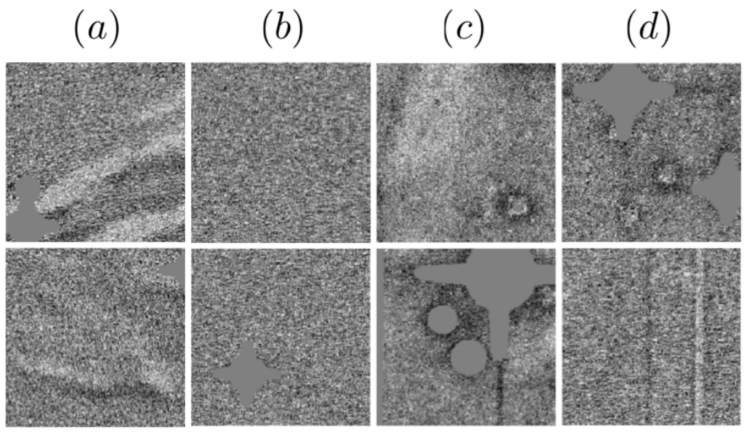


Figure 3.3: Examples of types of entities present in the CFHT dataset. Column (a) is of images that clearly contain at least one light echo, column (b) is of images that clearly do not contain at least one light echo, column (c) is of images that contain light echoes and artifacts, and column (d) is of images that contain entities which have similar characteristics to light echoes.

stellar dust has been illuminated by the brief outburst of a supernova. The portions of the dust that are illuminated change with time and therefore the illumination appears to move relative to other objects (like stars). Such locations are highlighted in difference images, see Figure 3.3 $(a), (c)$. To produce difference images, two exposures are obtained at the same telescope pointing (i.e. same celestial coordinates) and then one is subtracted from the other (after compensating for differences in depth and image quality.) Difference imaging can also produce a variety of artifacts, some of which are objects that mimic the characteristics of light echoes [12]. Artifacts can be caused by but are not limited to effects of telescopic pointing offsets, light from bright stars scattering of optics and the imager, diffraction, and cosmic rays. see Figure 3.3 $(c), (d)$. It is of interest to find difference images that contain entities that have a high probability of being a light echo, and to localize such entities since visual examination of large numbers of difference images is far too time-consuming. Supernovae light echoes reveals themselves in difference images as roughly equally-bright positive and negative flux diffuse features, displaced by tens of pixels in a given direction.

Approximately 13,000 2048×4612 difference images were produced from the survey and manually inspected over the course of one year, of which only 22 were found to contain at least one supernova light echo [12]. The 22 light echo containing CFHT difference images of size 2048×4612 were reduced to 350 difference images of size 200×200, among which 175 contained at least a portion of a light echo and the remaining 175 contained other astronomical entities. The dataset was created by manually masking the light echoes present in the 22 images of size 2048×4612, which were then cropped to size 200×200. If a cropped image contained at least 2500 pixels of mask then it was classified as containing a light echo. Among the 4885 200×200 cropped images that did not contain a light echo, 175 were selected at random, along with 175 that were classified as containing a light echo to form the final dataset of 350 images. The dataset consists of images and their respective binary labels, where the label indicates whether the image does or does not contain at least one light echo. The dataset was split into a training set of 250 images, and a validation and test set of 50 images each. The dataset was not augmented with rotations and transformations of the original cropped images because the part-whole relationship learned by capsule networks allows for them to generalize well to novel viewpoints [5].

A diagram of the capsule network architecture is given in Figure 3.5, and specified in Table 3.1. To elaborate, a 200×200 image is convolved with a set of eight 9×9 filters using a stride of three to produce the feature map. The feature map is convolved with a set of sixteen 5×5 filters using a stride of 2 to produce the ConvCaps layer. The subsequent capsule layers contain 8, 3, and 1 capsules of dimension 6, 8, and 8, respectively. The network is trained such that the length of the single capsule in capsule layer 3 is approximately 1 when at least one supernovae light echo is present in the image, and approximately 0 otherwise. This is achieved by minimizing the margin loss function, L, and given by

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

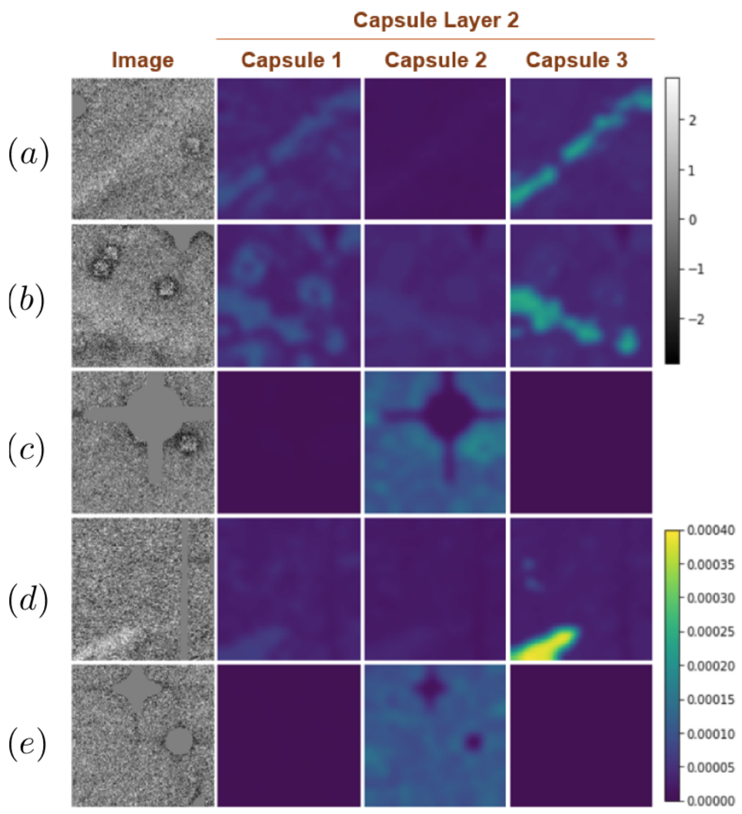


Figure 3.4: Routing path visualization to interpret the entities that the capsules in capsule layer 2 detect. Sample images from test set. Sample test images are of size 200×200, and routing path visualization images of capsule layer 2 are of size 29×29. Images (a), (b), and (d) contain light echoes; (c) and (e) contain stars. All images show varying amounts of interstellar dust or artifacts.

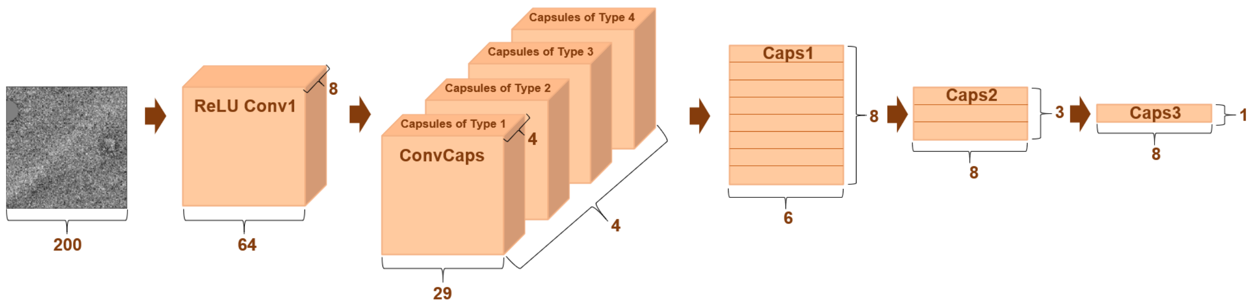


Figure 3.5: The capsule network architecture that was trained using the CFHT dataset.

where T=1 if at least one light echo is present in the image, m^+=0.9, $||\mathbf{v}||$ is the length of the capsule in capsule layer 3, λ=0.5, and m^-=0.1 [14].

Routing path visualization was used on the test set to interpret the entities that were detected by the capsules in the second capsule layer. Figure 3.4 reveals how much of each region in an image is routed to each of the three capsules on a sample of 5 images from the test set. These results are representative of the results on all 50 images in the test set. The brighter the region in the routing path visualization image, the more that region is routed to that capsule. Light echoes appear brighter and darker than the (zero-mean) background, but do not have a specific shape. When groups of supernova light echoes are seen, they generally are linearly distributed on the difference image. It is clear that capsule 3 detects light echoes as it precisely localizes them from the image. When a light echo is not present in the image, the routing path visualization image for capsule 3 is mostly empty. This is interesting as the network was trained using only images and their respective class labels. Typically, a more supervised approach is required to train a network for object localization, such as, providing the set of bounding boxes or pixel coordinates that contain the object [13].

Capsule 2 seems to be detecting the difference in brightness around bright stars, as seen in Figures 3.4 (c) and (e). The routing path visualization image is mostly empty when a bright star is not present in the image, as seen in Figures 3.4 (a), (b), and (d). Capsule 1 seems to be detecting a combination of light echoes and artifacts, as seen in Figures 3.4 (a) and (b). The visualization in Figure 3.4 suggest that the model may fail when given difference images containing bright stars. In other words, all capsules in capsule layer 2 are routed to the single capsule in capsule layer 3 which should be detecting light echoes. Since the light from bright stars is being routed to the light-echo-detecting capsule, the light-echo-detecting capsule may use that information to incorrectly make a prediction.

3.6 Conclusions

It is of great importance to understand how capsule networks make predictions, especially when the application concerns human well-being, in order to anticipate where the network may fail. Recent work on capsule network interpretation focused on understanding what features the individual dimensions of a capsule encode [14]. In this work, we have shown that routing path visualization can be used to interpret the entities that intermediate and final layer capsules detect, determine situations when the network may fail, and precisely localize the predicted class from the image. Weight sharing allows for capsule networks with a small number of weights to be created, thereby facilitating capsule networks to be trained on small datasets without over-fitting.

For future work, a decoder network [14] that reconstructs the image from an intermediate capsule layer can be attached to the architecture, and hence, an additional reconstruction loss will be added to the margin loss to form the total loss. The reconstruction loss should act as a regularization term, and as a consequence, improve model performance. It may be interesting to see how routing path visualization performs on a capsule network that has a decoder network. Furthermore, the capsule network architectures that were tested contained only one ConvCaps layer. It would be interesting to see how having more ConvCaps layers will affect the routing path visualization image.

3.7 Acknowledgements

This research was enabled in part by support provided by Compute Ontario (www.computeontario.ca) and Compute Canada (www.computecanada.ca). The images analyzed were based on observations obtained with MegaPrime/MegaCam, a joint project of CFHT and CEA/DAPNIA, at the Canada-France-Hawaii Telescope (CFHT) which is operated by the National Research Council (NRC) of Canada, the Institut National des Sciences de l'Univers of the Centre National de la Recherche Scientifique of France, and the University of Hawaii.

Chapter 4

A Package for the Automated Classification of Images Containing Supernova Light Echoes

Amanjot Bhullar[*], R. Ayesha Ali[*], Douglas L. Welch[a]

[*]Department of Mathematics and Statistics University of Guelph, Canada

[a]Department of Physics and Astronomy McMaster University, Canada

4.1 Abstract

The light echoes of supernovae can be detected in astronomical images; however, light echoes are extremely rare which makes manual detection a strenuous task. The aim is to develop an algorithm that can automatically localize the regions in an image that have a high probability of containing a supernova light echo. A capsule network is trained to automatically classify an image as containing or not containing at least one supernova light echo [14], and routing path visualization is used to localize light echoes in images [1]. A capsule network model was found

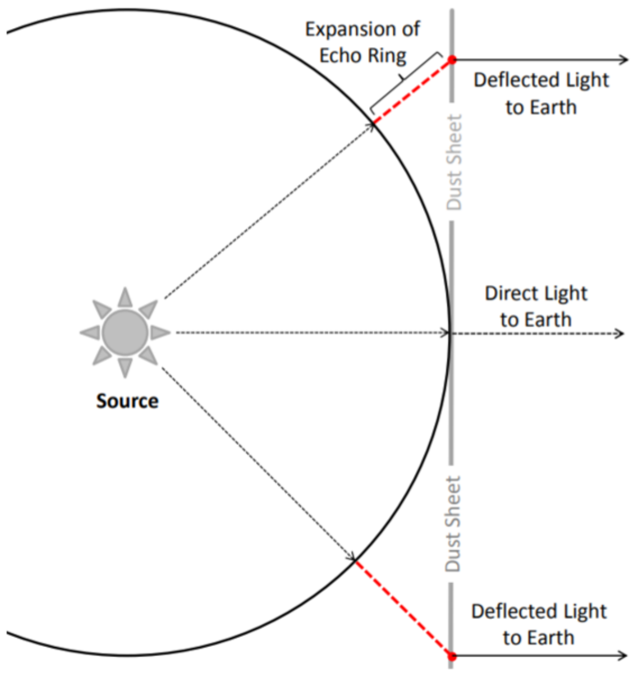


Figure 4.1: Light from a supernova travels radially outwards until it is deflected by interstellar dust. Deflected light is referred to as a light echo. Illustration credit: [12].

to achieve 90% classification accuracy on the test set, and experimental results demonstrate that light echoes can be precisely localized when routing path visualization is applied to the model. The package is called ALED and is available via github.com/AmanjotBhullar/ALED.

4.2 Introduction

When a star explodes at the end of its life, known as a supernova, the bright light from the explosion is scattered by interstellar dust as it moves outwards, see Figure 4.1. The deflection of light off interstellar dust is analogous to the deflection of sound waves off surfaces; hence the term light echo. Light echoes can arrive to Earth centuries post explosion, thus, light echoes facilitate the study of historic supernovae in modern times. The light echoes of supernovae are faintly visible in astronomical images, but are rare and difficult to detect against the background of space [12].

A difference image is the result of subtracting a pair of images that are taken at the same telescopic pointing but at different dates. Difference imaging allows for objects that move relative to the background of space, such as light echoes, to be manually detected more easily (see Figure 4.2). Difference imaging was originally implemented to speed up the manual analysis process. However, manual image analysis is still demanding due to the large amount of data generated by telescopes. For instance, in McDonald (2020) 13,000 difference images were manually inspected for light echoes over the course of a year.

This paper introduces a Python package, called ALED, for the automated detection of light echoes in difference images. The package takes as input a black and white difference image of arbitrary size, and outputs a corresponding routing path visualization image of the input image. The routing path visualization image reveals the regions of the input image that have a high probability of containing a light echo, and consequently, localizes the light echoes [1]. ALED's output image is useful as it allows astronomers to quickly identify the areas of the difference image that resulted in the image being classified as containing at least one light echo. Interpreting how a model functions is important because it will allow for the identification of scenarios that may result in model failure. This is especially important when the training set is suspected to not capture difficult image classification scenarios that are likely to arise in the future, which is an issue for small training sets. Consequently, constraints can be applied to the network, or improvements can be made to the training set to decrease the probability of failure. Model 6 can classify a 200×200 difference image in 0.63 seconds (Table 4.3), and a corresponding routing path visualization image can be produced in approximately 2 seconds.

Section 4.3 describes the real world astronomical data used to train ALED. Section 4.4 briefly reviews capsule networks and presents the model implemented in ALED. Section 4.5 compares performances to several capsule network and convolutional neural network (CNN) architectures on the CFHT dataset. Concluding comments are made in Section 4.6.

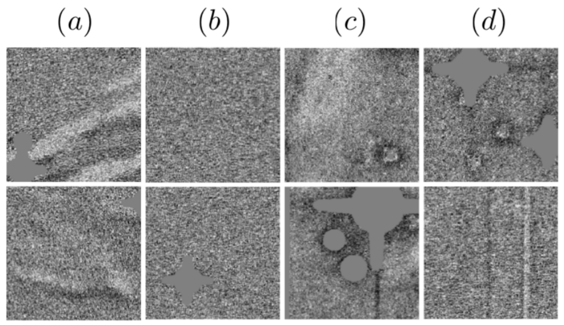


Figure 4.2: Examples of types of entities present in the CFHT dataset. Column (a) is of images that clearly contain at least one light echo, column (b) is of images that clearly do not contain at least one light echo, column (c) is of images that contain light echoes and artifacts, and column (d) is of images that contain entities which have similar characteristics to light echoes. Illustration and caption credit: [1]

4.3 CFHT Dataset

In 2011, the Canada-France-Hawaii Telescope (CFHT) MegaCam conducted a survey, with the primary objective of discovering supernova light echoes. Out of the 13,000 2048×4612 difference images that were produced from the survey, 22 were found to contain at least one light echo. The data from the survey was later reduced to a dataset, referred to as the CFHT dataset, of 350 200×200 images of which half contained at least a portion of a light echo, and the remaining half were of other astronomical entities. The CFHT dataset was not augmented with rotations and transformations as the technique is deemed unnecessary as capsule networks learn viewpoint invariant knowledge [?]. The data reduction process is explained in Section 4.2 of [1]. The CFHT dataset was split into a training set of 250 images, and a validation and test set of 50 images each.

Difference imaging may also create artifacts that mimic the shape and brightness of light echoes, see Figure 4.2. Simpler automatic image classification techniques struggle to distinguish light echoes from light-echo-like artifacts [12].

4.4 Classifier

CNNs have achieved state-of-the-art performance on many computer vision problems, and are currently one of the most popular algorithms for image classification. CNN architectures typically contain millions of weights that are to be learned during training, and as a result, require a large training set to prevent over-fitting [11]. However, in specialized fields, such as observational astronomy, sufficiently large sets of labeled data are often unavailable. This is especially true for automating the classification of images containing supernova light echoes because supernova are extremely rare events [12]. Capsule networks typically contain fewer trainable weights than CNNs, and as a result, require a smaller training set to achieve good performance [14]. Capsule networks also facilitate routing path visualization which can be used to interpret the entity that a given capsule detects.

A capsule is a vector whose elements represent the instantiation parameters of an entity in the image, where an entity is defined as an object or object part. The instantiation parameters of an entity are defined as the total information that would be required to render the entity. For example, the instantiation parameters of a circle whose center is positioned at coordinates (x, y) could be given by (x, y) and the radius of the circle. A capsule network consists of multiple layers of capsules, where a capsule in a given layer detects a particular entity. Capsules in a layer are children to the capsules of its succeeding layer, and parent to the capsules of its preceding layer. The child-parent coupling of some pairs of capsules is stronger than others, and this is determined by the routing algorithm, as explained in [14].

Initially, when an image is fed into a capsule network, the image is convolved with a set of filters to produce a feature map, i.e. to produce a tensor representation of the image that is better suited for classification [4]. The size of the feature map is determined by the length, F, and stride, S, of the filters, and the total number of filters used, N. Feature maps can be produced from feature maps, which implies that a network may have M feature maps in total.

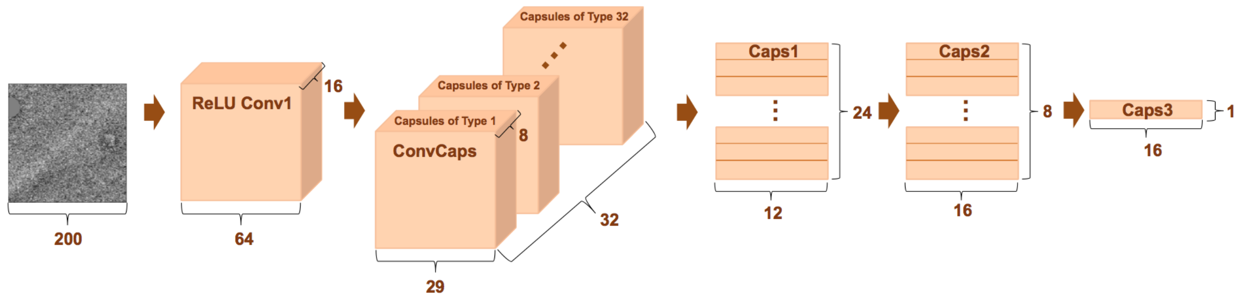


Figure 4.3: A diagram of the architecture of model 6 in Table 4.1.

The M^{th} feature map is convolved with a set of filters to produce a convolutional capsule (ConvCaps) layer. The ConvCaps layer contains I capsule types each of dimension D, where all capsules of a particular type detect the same entity. The elements of a capsule in the ConvCaps layer are calculated using local regions of the image, so ConvCaps capsules detect simpler entities. Succeeding layer capsules are calculated using preceding layer capsules and transformation weight matrices. Hence, higher-level capsules look at broader regions of the image, and are expected to detect more complex entities.

Each capsule in the final capsule layer is forced to detect a particular entity, such as a light echo, whilst intermediate layer capsules are not specified to which entities they may detect. The Euclidean length of a capsule will be close to 1 if the entity that the capsule detects is present in the image and close to 0 otherwise. This phenomenon is enforced by minimizing the margin loss while training,

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

where $T{=}1$ if the entity is present in the image, $m^{+}{=}0.9$, $||\mathbf{v}||$ is the length of the capsule, $\lambda{=}0.5$, and $m^{-}{=}0.1$ [14]. The total margin loss is calculated by summing the margin loss of each capsule in the final capsule layer. If the Euclidean length of the capsule is close to 1, it is said to be "active".

The architecture of the capsule network model that is available in ALED to classify images as containing or not containing at least one light echo is shown in Figure 4.3, and labeled as model 6 in Table 4.1. To elaborate, a 200×200 image is convolved with a set of 16 filters sized 9×9 using a stride of 3 to produce the feature map. The feature map is convolved with a set of 256 filters sized 5×5 using a stride of 2 to produce the ConvCaps layer. The subsequent capsule layers contain 24, 8, and 1 capsules of dimension 12, 16, and 16, respectively. The dimensionality of higher-level capsules is larger because more degrees of freedom are required to store the instantiation parameters of more complex entities. The length of the capsule in capsule layer 3 will be close to 1 if a light echo is present in the image, and 0 otherwise. ALED also uses weight sharing in which capsules of the same type within a ConvCaps layer share a set of transformation matrices [1].

The models were trained using a single NVIDIA Pascal P100 GPU on TensorFlow [2] with the Adam optimizer [7]. To train the capsule network model, the learning rate was initially set to 0.001 because it was found to be the largest learning rate that caused a steady decline in the total margin loss. Moreover, a batch size of 5 was used because it was the largest size that could fit into memory. While training, if the total margin loss of the validation set had not decreased for 10 consecutive iterations then the learning rate was decreased by a factor of 10. Learning was terminated when the learning rate fell below 10^{-6}.

The image was classified as containing a light echo if the length of the final capsule was greater than 0.5. The accuracy of the trained model was defined by

$$\frac{\texttt{True Positives}+\texttt{True Negatives}}{\texttt{No. of Images in Set}}\times 100\%,$$

and was found to be 90% on the test set and 88% on the validation set.

4.5 Results

4.5.1 Model Architectures

Eleven capsule network models and five CNN models were trained on the CFHT dataset, as summarized in Tables 4.1 and 4.2, respectively. The 16 models were trained using the same initial learning rate, batch size, and training script per Section 4.4. The accuracy and time taken to classify a single image is summarized in Table 4.3 for each of the 16 models. The CNN architecture in Table 4.2 served as baseline comparisons for the classification accuracy of the capsule networks. A larger augmented training set is typically required to train a CNN architecture, as the model quickly over-fits to small training sets of size 250 and 500 images. Models 13, 14, and 15 used the same CNN architecture, but models 13 and 14 were trained using an augmented training set of 1000 and 500 images, respectively. The original training set of 250 images was flipped horizontally to create the augmented set of 500 images, and then flipped vertically to create the augmented set of 1000 images in total.

Training was terminated just before the model began over-fitting to the training set. Due to the large number of weights in the CNNs and the small training set, the two fully connected layers before the output layer required a 95% dropout to combat over-fitting. Since dropout is typically set to 50% [6], it may be that too many neurons were initialized in the fully connected layers thereby resulting in strong over-fitting. Model 16 was trained on the augmented training set of 1000 images to see how a CNN model with less than one million trainable weights, and a more standard percent dropout of 50% would perform.

4.5.2 Results

For the capsule network models, it was found that classification accuracy and time taken to classify an image increased as the number of capsule layers increased (Tables 4.1 and

	Convolutional Layers			**Capsule Layers** (C, D)			
Model	**Number F. Maps**	**Feature Maps** $M \times (F, N, S)$	**ConvCaps** (F, I, D, S)	1	2	3	**Total Weights**
1	5	3 × (5, 256, 1) 2 × (5, 256, 2)	(5, 40, 12, 2)	(30, 15)	(1, 25)		9,861,010
2	2	2 × (5, 256, 2)	(5, 40, 12, 2)	(30, 15)	(1, 25)		4,945,042
3	2	2 × (5, 256, 2)	(5, 40, 12, 2)	(1, 15)			4,724,992
4	1	1 × (9, 256, 3)	(5, 40, 12, 2)	(30, 15)	(10, 25)	(1, 25)	3,428,222
5	1	1 × (9, 256, 3)	(5, 40, 12, 2)	(30, 15)	(1, 25)		3,320,722
6	1	1 × (9, 16, 3)	(5, 32, 8, 2)	(24, 12)	(8, 16)	(1, 16)	216,608
7	1	1 × (9, 16, 3)	(5, 24, 8, 2)	(16, 10)	(4, 12)	(1, 12)	117,280
8	1	1 × (9, 16, 3)	(5, 6, 4, 2)	(8, 6)	(4, 8)	(1, 8)	13,880
9	1	1 × (9, 8, 3)	(5, 4, 4, 2)	(8, 6)	(3, 8)	(1, 8)	5,984
10	1	1 × (9, 8, 3)	(5, 2, 4, 2)	(6, 4)	(2, 6)	(1, 6)	2,816
11	1	1 × (9, 8, 3)	(5, 2, 3, 2)	(6, 4)	(3, 6)	(1, 6)	2,546

Table 4.1: List of capsule network architectures trained on the CFHT dataset. Models are defined by the feature map(s), ConvCaps layer, and capsule layer(s), where C is the number of capsules in the layer, D is the number of dimensions per capsule, F is the length of each filter, N is the number of filters used, S is the stride, I is the number of capsule types, and M is the number of feature maps. The Number F. Maps column lists the total number of feature maps in each model. The Total Weights column lists the total number of trainable weights in each model. All models trained on 250 images.

			Fully Connected (FC) Layers				
Model	**Train Size**	**Number F. Maps**	**Feature Maps** $M \times (F, N, S)$	**Number FC Layers**	**Number Neurons**	**Dropout %**	**Total Weights**
12	250	3	2 × (5, 256, 2)	3	328	0	22,848,778
			1 × (5, 128, 2)		192	50	
					2	-	
13	1000	3	1 × (9, 256, 3)	3	152	95	4,551,906
			2 × (5, 128, 2)		88	95	
					2	-	
14	500	3	1 × (9, 256, 3)	3	152	95	4,551,906
			2 × (5, 128, 2)		88	95	
					2	-	
15	250	3	1 × (9, 256, 3)	3	152	95	4,551,906
			2 × (5, 128, 2)		88	95	
					2	-	
16	1000	3	1 × (9, 16, 3)	3	152	50	875,586
			2 × (5, 32, 2)		88	50	
					2	-	

Table 4.2: List of CNN architectures trained on the CFHT dataset. Models are defined by the feature maps, fully connected layers, and % dropout per fully connected layer, where F is the length of each filter, N is the number of filters used, S is the stride, and M is the number of feature maps. The Number F. Maps column lists the total number of feature maps in each model. The Total Weights column lists the total number of trainable weights in each model.

			Accuracy		Training		
Network Type	**Model**	**Weights**	**Validation**	**Test**	**Duration**	**Steps**	**Time** (s)
Capsule	1	9,861,010	80%	78%	14h	7860	-
	2	4,945,042	84%	86%	15h	7692	0.35
	3	4,724,992	50%	50%	2h	11990	0.013
	4	3,428,222	88%	88%	11h	2965	0.79
	5	3,320,722	82%	82%	11h	3250	0.83
	6	216,608	88%	90%	4h	1620	0.63
	7	117,280	88%	92%	2h	1950	0.22
	8	13,880	88%	90%	< 1h	3960	0.022
	9	5,984	88%	92%	< 1h	9200	0.015
	10	2,816	84%	90%	< 1h	2320	0.0085
	11	2,546	86%	84%	< 1h	2015	0.0086
Convolutional	12	22,848,778	Over-Fitting		< 1h	456	-
	13	4,551,906	88%	86%	< 1h	1653	0.0029
	14	4,551,906	82%	86%	< 1h	360	0.0029
	15	4,551,906	60%	66%	< 1h	234	0.0029
	16	875,586	66%	60%	< 1h	1000	0.0016

Table 4.3: The prediction accuracy on the validation and test sets is listed for each model. The time taken to train each model is given under the Duration column. The total number of times the weights were updated during training is given under the Steps column. The time taken to classify a single image is given under the Time column, and this number is averaged over 50 images.

4.3). Increasing the number of feature maps, M, and the size of the ConvCaps layer did not noticeably affect the classification accuracy. The total number of trainable weights in a model was mainly influenced by the size of the feature map(s) and ConvCaps layer, and not the number of capsule layers. A notable difference between capsule network and CNN models is that capsule networks can be trained to achieve good classification accuracy using a small non-augmented training set, and an architecture that contains a few thousand trainable weights.

The entity that a given capsule detects can be localized, if it is present, by routing path visualization [1]. Hence, this technique can be used to interpret the entities detected by intermediate layer capsules. In turn, scenarios can be identified that may result in model failure. For instance, there is one capsule in capsule layer 3, and as a result, the information present in all active capsules in the preceding layer, capsule layer 2, will be strongly routed to that single capsule [14]. Consequently, if a capsule(s) in capsule layer 2 is detecting an irrelevant object, such as an artifact, than it may adversely influence predictions made by the light-echo-detecting capsule. However, changes to the training set or to the model can help ensure that all capsules are detecting sensible objects.

Figure 4.4 displays sample routing path visualization images of the capsules in capsule layer 2 for model 9 (Table 4.1). The brighter the region in a routing path visualization image the more that region of the image is routed to that particular capsule. Capsule 1 seems to be detecting a combination of light echoes and artifacts; capsule 2 seems to be detecting the light around bright stars; capsule 3 seems to be detecting light echoes. Accordingly, the model may fail when given an image of a bright star because capsule 2 will become active, and the irrelevant information contained in capsule 2 will be routed to the light-echo-detecting-capsule. It was found that increasing the number of capsule types, I, and the number of dimensions per capsule, D, resulted in a cleaner routing path visualization image. Even though models 6, 7, 8, 9 have comparable classification accuracies (Table 4.3), model

6 was chosen to be the most ideal model because the routing path visualization image that it produces for the light-echo-detecting-capsule localizes light echoes with the most precision (Figure 4.6 (ii)).

Model 6 has a similar architecture to model 9, but contains more filters, capsules, and dimensions per capsule. The absence of artifacts in the routing path visualization images in Figure 4.5 reveals that the capsules in capsule layer 2 of model 6 are not nearly as sensitive to artifacts as model 9. This finding is reasonable because model 6 is more complex than model 9, and thus, can learn a more appropriate strategy for classification. All capsules in capsule layer 2 seem to be detecting entities related to light echoes, which makes the model less susceptible to incorrectly classifying a non-light-echo image as a light echo image.

Since there is only a single capsule in capsule layer 3, the routing path visualization of that capsule will simply be an addition of the routing path visualization images of the capsules in the preceding layer, capsule layer 2. From Figure 4.6, it is evident that a routing path visualization image of the light-echo-detecting capsule in model 6 precisely localizes light echoes, if they are present in the image. On the contrary, a routing path visualization image of the light-echo-detecting capsule in model 9 localizes other entities in addition to light echoes. It seems that a model with more weights can more cleanly localize light echoes in a routing path visualization image. In short, model 9 does not always use the correct information in the image to predict the presence of a light echo because irrelevant regions of the image are routed to the light-echo-detecting capsule. It is important to note that CNN models cannot produce a routing path visualization image as this is specific to capsule networks. ALED is comprised of a capsule network model, referred to as model 6 in Tables 4.1 and 4.3, and described in Section 3.

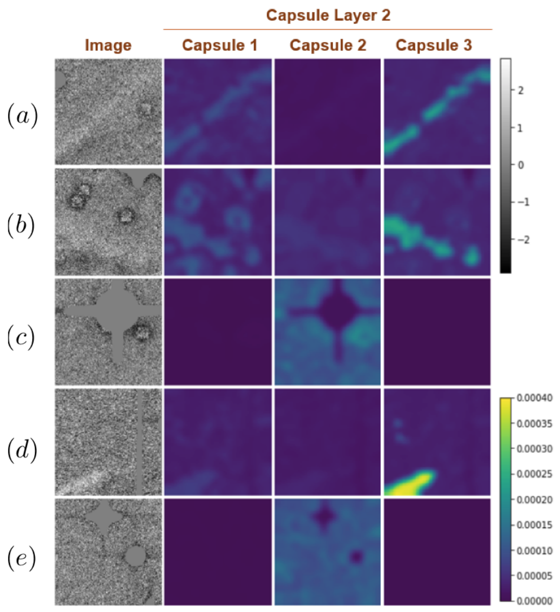


Figure 4.4: Routing path visualization to interpret the entities detected by the capsules in capsule layer 2 of model 9. Sample images from test set. Sample test images are of size 200×200, and routing path visualization images of capsule layer 2 are of size 29×29.

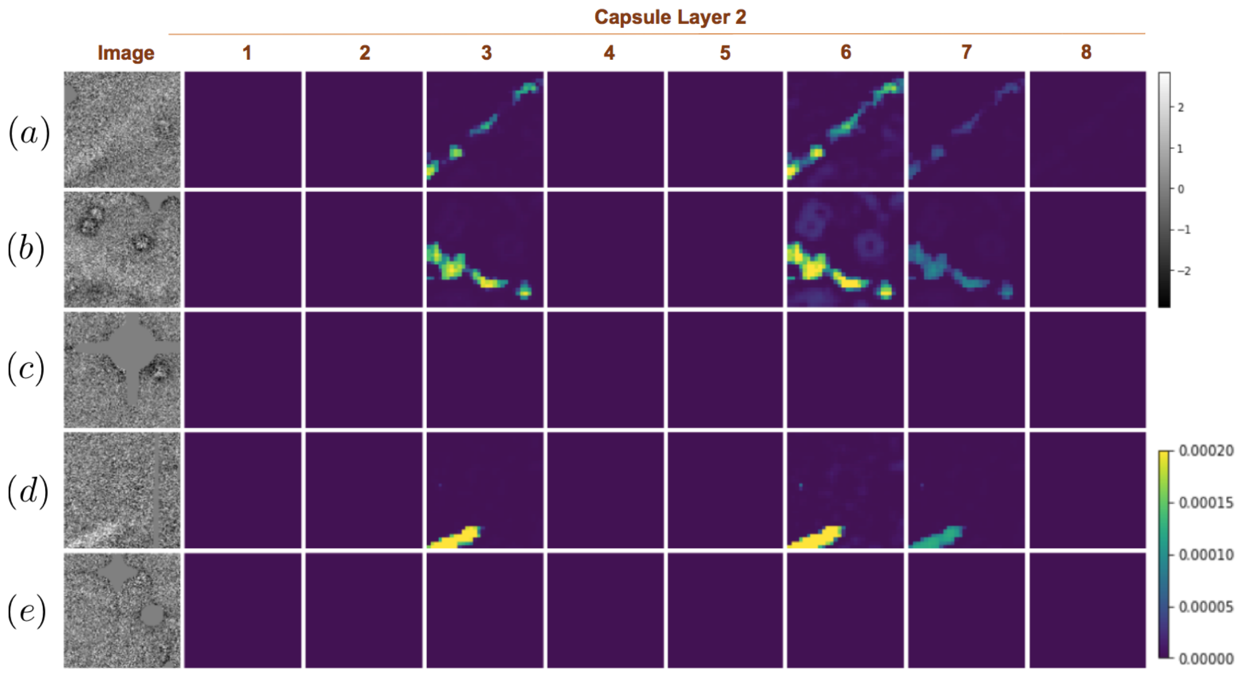
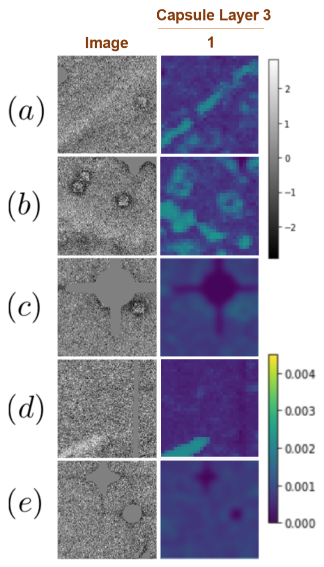
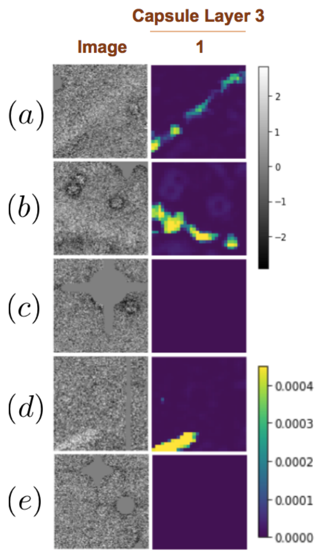


Figure 4.5: Routing path visualization to interpret the entities detected by the capsules in capsule layer 2 of model 6. Sample images from test set. Sample test images are of size 200×200, and routing path visualization images of capsule layer 2 are of size 29×29.



(i) Model 9



(ii) Model 6

Figure 4.6: Routing path visualization to localize light echoes detected by the light-echo-detecting-capsule. Sample images from test set. Sample test images are of size 200×200, and routing path visualization images of capsule layer 3 are of size 29×29.

4.6 Summary

A small dataset of 350 images was constructed from difference images manually identified to contain supernova light echoes in McDonald (2012). Several capsule network and CNN models were trained and tested using the dataset. It was found that CNNs and capsule networks have similar classification performance. Nevertheless, capsule networks still seem more favorable because of there added ability to localize detected entities for free via routing path visualization.

Capsule network model 6 was found to be using the correct regions of an image when predicting the presence of a light echo, whereas model 9 was not. In addition, model 6 was able to precisely localize light echoes when they were present in an image. The Python package ALED implements model 6, trained on the CFHT data, to predict the presence of light echoes in astronomical images. In particular, the function `classify_fits()` takes as input the path of a directory containing difference images, of any size, to be classified. Each difference image is cropped to many images of size 200×200, which are then passed to model 6 to be classified and a corresponding routing path visualization image is produced. The routing path visualization images are stitched together to form a final output routing path visualization image that corresponds to the input image. The routing path visualization image localizes the light echoes for the user. In addition, a text file is produced listing images that are good candidates for containing a light echo. This is the first automation of this laborious task that the authors are aware of.

4.7 Acknowledgements

This research was enabled in part by support provided by Compute Ontario (www.computeontario.ca) and Compute Canada (www.computecanada.ca). The images analyzed were based on observations obtained with MegaPrime/MegaCam, a joint project of CFHT and CEA/DAPNIA, at the Canada-France-Hawaii Telescope (CFHT) which is operated by the National Research Council (NRC) of Canada, the Institut National des Sciences de l'Univers of the Centre National de la Recherche Scientifique of France, and the University of Hawaii.

Chapter 5

Conclusion And Future Work

The contributions of this thesis are three fold

- Routing path visualization is introduced for interpreting capsules from a capsule network
- Performance of capsule networks is compared to convolutional neural networks for light echo classification
- The Python package ALED is developed for astrophysicist to automate light echo detection from astronomical difference images.

The results of this thesis provided several insights into how capsule networks behave and can be improved. For instance, the entities detected by intermediate layer capsules were interpreted, and the capsule network model was identified to possibly fail when given images containing bright stars or artifacts. In some instances this contamination may not pose a serious problem as there may be secondary interest in bright stars or artifacts, themselves. Regardless, it is expected that users would manually verify the presence of light echoes in the light-echo positive images, which is likely to be vastly smaller than the original set of images (e.g. McDonald surveyed 13,000 images but only 22 actually contained light echoes).

Routing path visualization was also shown to precisely localize the predicted class from an image when performed on capsules in the final layer. For example, the digits in the MNIST images were localized by applying routing path visualization on the appropriate digit-detecting capsule. While light echoes in the CFHT dataset were localized by applying routing path visualization on the light-echo-detecting capsule. Although CNNs are considered to be state-of-the-art for computer vision tasks, they require too much data to train for some real world applications and are prone to overfit. Capsule networks are found to perform well on astronomical images using fewer weights and without the need for data augmentation.

For future work, it can be tested how well routing path visualization performs when there is more than one ConvCaps layer in the network, or when a decoder network is attached to the capsule network.

Bibliography

[1] A. Bhullar, R. A. Ali, D. L. W. Interpreting capsule networks for image classification by routing path visualization. In *In preparation for the Journal of Machine Learning Research* (2020).

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] Brad Templeton. Tesla Bets Farm On Neural Network Based Autonomy With Impressive Presentation. https://www.forbes.com/sites/bradtempleton/2019/04/22/tesla-bets-farm-on-neural-network-based-autonomy-with-impressive-presentation/52732dd363ce, 2019. Online; accessed 21 March 2020.

[4] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[5] Hinton, G. E., Krizhevsky, A., and Wang, S. D. Transforming auto-encoders. In *International conference on artificial neural networks* (2011), Springer, pp. 44–51.

[6] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[7] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Kremer, J., Stensbo-Smidt, K., Gieseke, F., Pedersen, K. S., and Igel, C. Big universe, big data: machine learning and image analysis for astronomy. *IEEE Intelligent Systems 32*, 2 (2017), 16–22.

[9] LaLonde, R., and Bagci, U. Capsules for object segmentation. *arXiv preprint arXiv:1804.04241* (2018).

[10] LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[11] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[12] McDonald, B. J. The search for supernova light echoes from the core-collapse supernovae of AD 1054 (crab) and AD 1181 (master's thesis), 2012. McMaster University.

[13] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.

[14] Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in neural information processing systems* (2017), pp. 3856–3866.

[15] Shahroudnejad, A., Afshar, P., Plataniotis, K. N., and Mohammadi, A. Improved explainability of capsule networks: Relevance path by agreement. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (2018), IEEE, pp. 549–553.

[16] U.S. Food and Drug Administration. Artificial Intelligence and Machine Learning in Software as a Medical Device. https://www.fda.gov/medical-devices/software-medical-device-samd/artificial-intelligence-and-machine-learning-software-medical-deviceregulation, 2020. Online; accessed 21 March 2020.