COMP 551 Final Project

Track 3: Reproducibility Challenge

Matrix Capsules with Dynamic EM Routing

Kristy Mualim 260679030 kristy.mualim@mail.mcgill.ca
Eva Suska 260618967 eva.suska@mail.mcgill.ca
Benjamin Taubenblatt 260626105 benjamin.taubenblatt@mail.mcgill.ca
Katy Dong 260610798 katherine.dong@mail.mcgill.ca

Abstract Image classification is a difficult task in Machine Learning, especially because images can drastically vary in characteristics such as lighting, shape, and rotation, to name a few. Recently, there have been advancements in image classification via neural networks through the use of matrix capsules. Capsules are groups of neurons which represent part whole relationships of the same entity. Lower level capsules "vote" for the pose matrix—or a 4x4 matrix which learns to represent relationships between capsules—of capsules in the next layer. Capsules vote by multiplying its own pose matrix by a trainable viewpoint-invariant transformation matrix. The transformation matrices are trained using backpropagation and predictions are made via agreement between capsules. The idea is to use a Gaussian mixture model to cluster data points into a mixture of Gaussians. Expectation Maximization tries to fit data points into these Gaussians. In Matrix Capsules with EM Routing [1], the authors create a capsule network using EM routing. They were able to reduce error rates by more than 45% on a realistic image dataset. The techniques described in this paper have the potential to improve image classification techniques. The results described in this paper suggest that if this technique is used to continuously improve neural networks, these techniques could be used in hospitals to recognize anomalies in medical imaging.

INTRODUCTION

The fundamental goal of this paper seeks to reproduce the results achieved by the paper; Matrix capsules with EM routing [1]. We aim to build the architecture and perform the EM routing algorithm described to reduce errors by 45% compared to the state of the art CNN on the smallNORB dataset.

The smallNORB dataset contains grayscale images of 50 toys belonging to 5 generic categories: four-legged animals, human figures, airplanes, trucks, and cars. The dataset consists of 24,300 images, at 6 different lighting conditions, 9 elevations (30 to 70 degrees every 5 degrees), and 18 azimuths (0 to 340 every 20 degrees) [2].

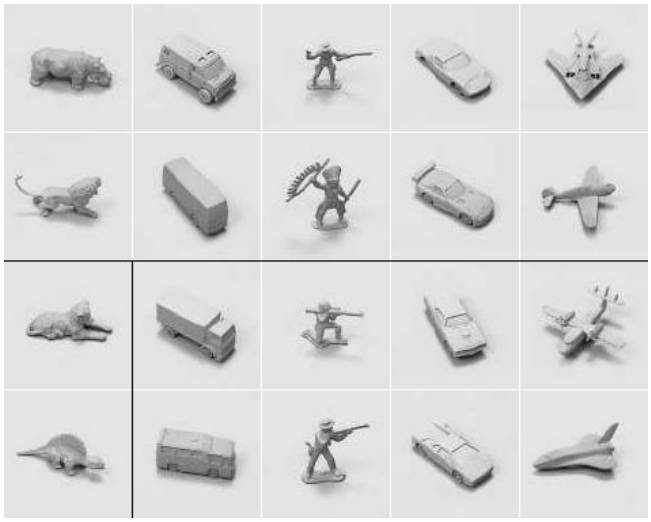

Fig. 1. Examples from the smallNORB dataset

In the following report, we will discuss the approach we took to build the final training model. We used python programming language with both the Tensorflow API and PyTorch. First, we performed the image preprocessing. Second, we implemented the algorithm as described in the paper. Lastly, we discussed the challenges we faced while implementing the models.

[1]

ALGORITHMS

A. *Baseline Convolutional Neural Network*

Convolutional Neural Networks (CNNs) form a class of deep, feed-forward artificial neural networks, usually applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing, indicating that the network learns the filters rather than having them be hand-engineered.

CNNs, similar to feed-forward neural nets, consist of an input and an output layer, with multiple hidden layers. Each individual hidden layer can be one of a convolutional, pooling, fully connected or normalization layers.

Convolutional layers apply a convolution operation to the input and passes the output as input to the next layer. Each convolutional neuron processes data only for its receptive field. The convolution operation applied helps to reduce the number of free parameters, allowing the network to be deeper in the second layer. After each convolutional layer, it is conventional to apply a nonlinear layer in an attempt to introduce nonlinearity to a system that was previously computing linear operations.

Pooling layers may be either local or global, both of which help to combine the outputs of neuron clusters at one layer into a single neuron in the next layer.

Fully connected layers connect every neuron in one layer to every neuron in another layer, its main principle is similar to a traditional multi-layer perceptron neural network.

CNNs are usually deemed more efficient than feed-forward neural networks in relation to image classification given its use of local receptive fields, parameter sharing and pooling. Local receptive fields help to focus on important features isolated to a small region of interest in the image. Parameter sharing is used by CNNs to control the number of parameters and is performed under the assumption that a feature useful to compute at a spatial position (x,y) would also be useful at a different position (x_2,y_2). Thus, neurons in each depth slice are fundamentally constrained to using the same weights and bias.

Training for CNNs is similar to that of feed-forward neural nets and involve backpropagation constituting a forward pass of training data, a loss function to evaluate error, a backward pass and a weight update.

B. *Capsules*

Instead of just a traditional neuron network, our network is composed of groups of neurons called capsules. Whie neural nets typically use simple non-linearities in which non-linear functions are applied to scalar outputs of a linear filter, capsules use a much more complicated non-linearity that converts whole sets of activation probabilities and poses of the capsules in a layer into activation probabilities and poses of capsules in the next layer. This specific use of forward propagating poses allows Capsules to be more robust in terms of viewpoint manipulation within image inputs.

A capsule network consists of several layers of capsules, where each capsule has is a 4x4 matrix called a pose matrix, whose entries represent information such as the x and y-coordinates of the feature, the angle of rotation of the object, and other characteristics. In addition, as well as an activation probability, a. The pose matrix of a given capsule is transformed by weights to case a respective vote for the pose matrix of the higher layers. These activation and pose matrices of all capsules in layer L+1 are calculated using a non-linear routing procedure.

Capsules outputs represents different properties of the same feature, where a single neuron outputs an activation from (0,1), and represents the existence of a feature, a capsule outputs a logistic unit (0,1) about the existence of a feature as well as a 4x4 pose matrix representing additional spatial information about the feature. The connection between two capsules is a 4x4 transformation matrix instead of a scalar weight in a regular neural network. The capsule gives us the ability to keep track and use the spatial information about a feature but the real effectiveness comes from the way the capsule information is transferred on by layer, in this case using EM Routing.

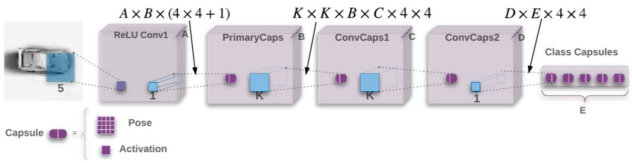

Fig. 2 Capsule Architecture

As illustrated in Fig 2, the general architecture proposed in this paper starts with a 5x5 convolutional layer with 32 channels (A=32) and a stride of 2 with ReLU non-linearity. Other layers are capsule layers starting with a primary capsule layer, where the 4x4 pose of each of the B=32 primary capsule types is a learned linear transformation of output of lower-layer ReLUs centered at a given location. The activations of primary capsules are generated by applying a sigmoid function to weighted sums of the same set of lower-layer ReLUs.

These primary capsules are later followed by two 3x3 convolutional capsule layers (K=3), each with 32 capsule types (C=D=32) with strides of 2 and one, respectively. The last layer of convolutional capsules is connected to the final capsule layer, where one capsule

illustrates an output class. Information is preserved when connecting convolutional capsule layers as well as the intrinsic nature of capsules of the same type extracting similar features at different positions. Information sharing is also present between different positions of the same capsule type, to which a coordinate addition is added. This adds a scaled coordinate (x,y) of the center of the receptive field of each capsule to the first two elements of the vote matrix, preserving positional information.

C. EM Routing

In statistics, a Gaussian mixture model clusters data points into a mixture of Gaussian distributions. The idea of using matrix capsules with Expectation Maximization (EM) routing is to cluster data points—or lower level capsules—into different Gaussians. Expectation Maximization (EM) tries to fit data points into these Gaussians. Clusters are created using maximum likelihood estimates (MLE).

The probability that a datapoint x belongs to cluster G_1 is

$$P(x|G_1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-(x-\mu_1)^2/2\sigma_1^2}$$

Higher level features in an image —e.g. a face— are calculated via agreement between votes of lower level capsules. A vote v_{ij} for the parent capsule j from capsule i is computed by multiplying the pose matrix M_i of capsule i with a viewpoint invariant transformation matrix W_{ij}.

$$v_{ij} = M_i \cdot W_{ij}$$

W_{ij} is learned discriminatively through a cost function and backpropagation. Even if the viewpoint changes, the pose matrices and votes will also change in a coordinated way. EM routing is based on proximity of votes and therefore, can still cluster the same data points together. Therefore the transformation matrices are the same for all viewpoints, making them viewpoint invariant.

Capsules are assigned to different higher level capsules via an assignment probability. For example, the hand capsule does not belong to the higher level face capsule and therefore would have an assignment probability of zero for that capsule.

Capsules are activated according to a cost function. The lower the cost, the more likely the capsule will be activated. If the cost is high, then the votes do not match the parent Gaussian distribution and therefore have a low chance to be activated.

Let v_{ij} be the vote from capsule i to parent capsule j and $v_{ij}^{\ h}$ be its h^{th} component. Applying the probability density function of a Gaussian distribution to a datapoint x, we get

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Computing the probability of $v_{ij}^{\ h}$ belonging to capsule j's Gaussian model, we get

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} \exp\left(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$$

Finally, taking the natural logarithm

$$\ln(p_{i|j}^h) = \ln \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} \exp\left(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$$

$$= -\ln(\sigma_j^h) - \frac{\ln(2\pi)}{2} - \frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}$$

Then the cost to activate parent capsule j from capsule i is

$$cost_{ij}^h = -\ln(P_{i|j}^h)$$

Capsule activations are determined using a variant of the sigmoid function.

$$a_j = sigmoid(\lambda(b_j - \sum_h cost_j^h))$$

In the paper [1], "$-b_j$" is described as being the cost of representing the mean and variance of capsule j. b_j is learnt through training via the cost function and backpropagation.

The pose matrix and the activation of the output capsules are computed iteratively using EM routing. Expectation Maximization fits data points into a mixture of Gaussian distributions using an E-step and an M-step.

The E-step determines the assignment probability r_{ij} of each data point to a parent capsule and the M-step re-calculates the Gaussian models' values based on r_{ij}. In the paper, the authors repeat these steps three times, keeping the last output as the parent capsules output. The whole algorithm is summarized as follows:

procedure EM ROUTING($\boldsymbol{a}, V$)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}, R, V, j$)
    $\forall i \in \Omega_L$: E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$)
  return $\boldsymbol{a}, M$

procedure M-STEP($\boldsymbol{a}, R, V, j$) ▷ for one higher-level capsule, j
  $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
  $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
  $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
  $cost^h \leftarrow \left(\beta_u + log(\sigma_j^h)\right) \sum_i R_{ij}$
  $a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$

procedure E-STEP($\mu, \sigma, \boldsymbol{a}, V, i$) ▷ for one lower-level capsule, i
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{(V_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}\right)$
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j p_j}{\sum_{k \in \Omega_{L+1}} \boldsymbol{a}_k p_k}$

Fig.3 EM Routing Algorithm

a and *V* above are the activation and votes from the children capsules respectively. The assignment probability r_{ij} is initialized to be uniformly distributed. In other words, at the start of the algorithm, we assume that the children capsules are equally related with any parent capsule.

First the M-step is called to compute an updated Gaussian distribution (μ, σ) and the parent activation a_j from *a*, *V*, and current r_{ij}.

Then the E-step is called to recompute the assignment probabilities r_{ij} based on the new Gaussian model and the new a_j.

Overall, EM routing is used to compute capsules' outputs.

D. Loss functions

Three different loss functions were used in the given paper, mainly spread loss, margin loss and cross-entropy loss. The main loss function being spread loss, with the other two acting as comparisons.

I. Spread Loss

Spread loss was utilized in the paper [1] to make training less sensitive to initialization and hyper-parameters of the model.

$$L_i = (max(0, m - (a_t - a_i))^2, \quad L = \sum_{i \neq t} L_i$$

Spread loss seeks to directly maximize the gap between activation of target (a_t) and the other classes. If the activation of a wrong class, a_i, is closer than margin, m, to a then it is penalized by the squared distance to the margin. The paper linearly increased the margin, starting from 0.2 up to 0.9 to avoid dead capsules in early layers. This loss function is equivalent to squared hinge loss where m=1.

II. Margin Loss

Margin loss, a loss function proposed by [3], is applied to each capsule, k:

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

where Tk=1 if a feature of class k is present and m+ = 0.9 and m- = 0.1. A regularization parameter is used to down-weight the loss for absent classes in order to prevent initial learning from shrinking the lengths of the activity vectors of all the digit capsules. Margin loss is used to allow for multiple feature representation within a capsule. The length of the instantiation vector is utilized to represent the probability that a capsule's entity exists and the presence of a top-level capsule for digit class k to have a long instantiation vector is dependent on if and only if the feature is present in the image. Total loss is illustrated as the sum of loss of all digit capsules.

III. Cross Entropy Loss

Cross Entropy Loss, or log loss, measures the performance of a classification model whose outputs are probability values between 0 and 1. Cross entropy seeks to provide an estimate for true distribution p(x) given two distributions over discrete variables x.

$$H(p, q) = -\sum_{\forall x} p(x) log(q(x))$$

METHODOLOGY

A. Data Preprocessing

Data Preprocessing was done utilizing the smallNORB dataset that consists of gray-level stereo images of 5 classes of toys: airplanes, cars, trucks, humans and animals. It consists of 10 physical instances of each class painted matte green. 5 physical instances of a class are selected for training data and the other 5 for test data. Each individual toy was pictured at 18 different azimuths (0-340), 9 elevations and 6 lighting conditions, causing training and test sets to each contain 24,300 pairs of 96x96 stereo images. The smallNORB was selected as a benchmark in developing this novel capsule matrices. The specifications of data preprocessing closely followed the details stated in the paper [1]. smallNORB images were downsampled to 48x48 pixels and normalized to have zero mean and unit variance. Training images consisted of cropped 32x32 patches and had the addition of random brightness and contrast to these cropped images. Test images were a 32 x 32 patch from the center of the image.

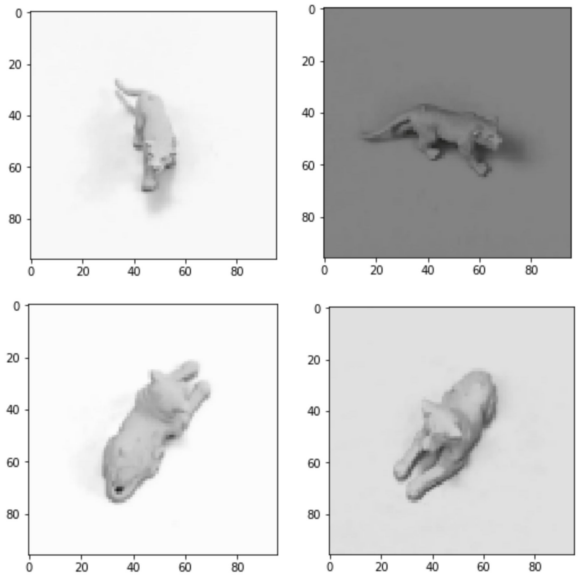

Fig 4: Original Image (left) with Preprocessed Image (Right). Preprocessed Images went through rotation and random contrast/brightness added.

B. Baseline CNN

As a baseline for our experiments, we trained a CNN according to the specification in the paper [1]. We have two convolutional layers one with 32 and and the next with 64 channels. Both layers have a kernel size of 5 and a stride of 1 with a 2 × 2 max pooling. The third layer is a 1024 unit fully connected layer with dropout and connects to the 5-way softmax output layer. All hidden units use the ReLU non-linearity. We also used the same image preprocessing as for the capsule network.

B. Capsules

As illustrated in Fig 2, we utilized the general architecture proposed in this paper that starts with a 5x5 convolutional layer with 32 channels (A=32) and a stride of 2 with ReLU non-linearity.

The primary capsule layer consisted of B=32 primary capsule types where the 4x4 pose of each of these layers represented a learned linear transformation of output of lower-layer ReLUs centered at a given location. The activations of primary capsules are generated by applying a sigmoid function to weighted sums of the same set of lower-layer ReLUs.

This was followed by two 3x3 convolutional capsule layers (K=3), each with 32 capsule types (C=D=32) with strides of 2 and one, respectively. The last layer of convolutional capsules is connected to the final capsule layer, where one capsule illustrates an output class.

We also incorporated Coordinate Addition applied in the paper as well as EM Routing by Agreement.

IV. CHALLENGES

We faced many challenges during this project, including preprocessing and implementing theoretical frameworks.

A. Data Preprocessing

Gathering the data from the respective NYU websites was relatively easy but we faced issues with data compatibility when it came to preprocessing. In addition, the given dataset contained 24,300 pairs of 96x96 pixel images, generating a right and left image in the data. It was confusing trying to determine the motivation behind having pairs of images saved, given that each image had only one set of corresponding details - azimuth, lighting, elevation, instance and category. In addition, utilizing tensorflow as a means for data processing resulted in the generation of tensor objects which was a different data type that we were not accustomed to. However, the use of tensorflow for data preprocessing was incredibly useful in data augmentation where, given in the presence of a larger dataset, the use of a GPU could seek to enhance matrix calculations and alter the different properties contained within each image such as altering azimuths and elevations.

B. Paper Ambiguity

The relative novelty and complexity of Matrix Capsules with EM Routing posed several challenges given certain obscurities present in the paper as well as our attempts at fully understanding how individual components were ran to produce the given results.

The paper [1] also heavily relies on the Expectation Maximization algorithm and its components, however, many of the variables and parts are not described in enough detail in order to really understand their intricacies. It would have been useful to us if the authors discussed the idea of EM routing being similar to a Gaussian mixture model. Without having previous experience with such models, we found ourselves using external resources in order to understand the role of each part of the EM routing algorithm described in the paper.

We also struggled with translating the original baseline CNN used in the paper into code. We tried using Tensorflow to implement the baseline CNN as outlined in the paper. However, we had some challenges while attempting to implement the CNN because some of the parameters required were not stated in the paper. For example, the third layer is a fully connected layer with dropout, but the probability of elements to be dropped was not included.

Histograms indicative of distances of votes to mean of 5 final capsules after each routing iteration in the

paper [1] did not follow with its methodology in generating these figures. An attempt was made to understand how these histograms were generated via viewing various implementations but it was often a conversation topic in many discussion posts.

As illustrated in the paper [1], a comparison of the smallNORB test error rate of the baseline CNN and the capsule model on novel viewpoints when both models were matched on error rate for familiar viewpoints was done. The test error illustrated 'Azimuth' and 'Elevation' as categories for both CNN and Capsules respectively, however, it was difficult understanding how these specific categories of test set were used to generate these error rates. Each given image has values relating to azimuth and elevation respectively.

C. Capsule Implementation

We initially took to tackling the paper utilizing a TensorFlow implementation and subsequently utilizing PyTorch.

One of the main challenges was creating the capsule network itself. The paper describes how capsules work from a theoretical perspective, but was rather difficult to implement utilizing both TensorFlow and PyTorch. Despite getting the general architecture of the capsule nets, we faced multiple implementation errors ranging from the utilization of new frameworks that we were not originally exposed to. In addition, the paper failed to properly illustrate and break down exactly what was done, such as the specific parameter and hyper-parameter values that were chosen to obtain the results obtained. Despite TensorFlow having documentation online, it was difficult to follow and execute the various methods as its documentation was primarily geared towards users who have had prior knowledge to utilizing such a framework. In addition to this, it was difficult to implement a capsule architecture based on these theoretical descriptions and without having experience building neural network variants from scratch. We were able to find support for capsule networks in the TensorFlow and TensorFlow-Slim libraries. These libraries provide support for structures such as the initial ReLu convolution capsule layers which are used to build other layers.

The subsequent PyTorch implementation allowed us to successfully build a capsule net framework with EM Routing. However, implementation was difficult in that we obtained issues pertaining to the PyTorch framework itself. Upon trying to resolve these issues, we noticed several open issues illustrating the problems we encountered that had yet to be resolved, with others being able to reproduce the issues we encountered. Due to these frameworks being a novelty, a lot of the time spent in trying to reproduce this paper was attributed to debugging and learning the different frameworks itself. In facing errors that we were not accustomed to, being resourceful in reaching out to previous implementations of a previously published paper [2] helped in resolving certain issues. In addition to this, the authors trained the model utilizing 8 sync gpus on the smallNORB dataset, which proved as hardware constraints in reproducibility. Without access to an already trained model, it would be difficult to obtain and replicate the results reported in the paper.

Overall, we learned a great deal about creating capsule network architectures from this project. We initially did not understand how tools such as TensorFlow and PyTorch could be utilized to help build such complex neural network implementations. However, after this project, we feel like we have created a strong base for implementing other theoretical papers using tools such as TensorFlow and PyTorch.

Conclusion

Improved image recognition techniques have many clinical applications and implications. Increased performance through the use of capsule networks and EM routing could improve image recognition techniques enough such that they can reliably be used in medicine. One such application is detecting cancer in mammography scans. Other applications of these techniques are increased performance and reliability in face detection software for security and detecting certain organisms in microscopic samples.

If the techniques described in this paper are studied more, the accuracy and reliability of these techniques may be stabilized enough to be used in critical software systems.

More research into capsule networks and EM routing must be done in order to improve these techniques enough to be able to use them reliably in sensitive models.

Future work could seek to extend this implementation via successfully utilizing either framework and providing a more in-depth implementation documentation for others to reproduce this paper with. In addition, we see this type of network being utilized in video imagery as well. In addition, the current implementation of capsule matrices require the input images to have the same height and width, an improvement could seek to allow this implementation on images that are alternatively-sized.

Summary of Contributions

The authors, Kristy Mualim, Katy Dong, Benjamin Taubenblatt, and Eva Suska, have taken roles in the implementation of the methodology, the evaluation, and the composition of this report.

Eva and Katy took on responsibilities tackling the baseline CNN models, Kristy took on responsibilities

related to preprocessing the data, building the capsule network architecture and running the EM routing algorithm, Benji took on responsibilities related to building the capsule network architecture and running the EM routing algorithm.

We all worked on various aspects of the report primarily related to the parts we were most knowledgeable about. We all contributed to the challenges and methodology sections, to name a few.

We hereby state that all the work presented in this report is that of the authors.

References

[1] G.E. Hinton, S. Sabour, and N. Frost, "Matrix capsules with em routing," 2018.

[2] F.J. Huang and Y. LeCun, "The small norb dataset v1.0", 2005. https://cs.nyu.edu/~ylclab/data/norb-v1.0-small/

[3] Sara Sabour, Nicholas Fross, and Geoffrey E Hinton. Dynamic routing between capsules. In *Neural Information Processing Systems (NIPS)*, 2017.