Learning to compute inner consensus - A noble approach to modeling agreement between Capsules*

Gonçalo Faria
Department of Informatics
School of Engineering
University of Minho

Abstract

This project considers Capsule Networks, a recently introduced machine learning model that has shown promising results regarding generalization and preservation of spatial information with few parameters. The Capsule Network's inner routing procedures thus far proposed, a priori, establish how the routing relations are modeled, which limits the expressiveness of the underlying model. In this project, we propose two distinct ways in which the routing procedure can be learned like any other network parameter.

1 Introduction

Starting with the developments made by Frank Rosenblatt surrounding the Perceptron algorithm [21], innovative techniques have marked the beginning of a new, biologically inspired, approach in Artificial Intelligence [20] that is, surprisingly, better suited to deal with naturally unstructured data.

The now called field of Deep Learning has expanded these ideas by creating models that stack multiple layers of Perceptrons. These Multilayer Perceptrons, commonly known as Neural Networks [7], achieve greater representation capacity, due to the layered manner the computational complexity is added, especially when compared with its precursor. Attributable to this compositional approach they are especially hard-wired to learn a nested hierarchy of concepts [27].

As an approach to soft-computing, Neural Networks stand in opposition to the precisely stated view of analytical algorithms that, unlike the human mind, are not tolerant of imprecision, uncertainty, partial truth and approximation [5]. In conjunction with other Deep Learning models, they stand at the vanguard of Artificial Intelligence Research, employed in tasks that previously have been found computationally intractable.

Aided by the increase in computational power as well as efforts to collect high quality labeled data, in the last decade, a particular subset of Neural Networks, called Convolutional Neural Networks(**CNN**) [16], have accomplished remarkable results [15]. As their common trait, having to deal with high dimension unstructured data, computer vision [15], speech recognition [10], natural language processing [3], machine translation [26] and medical image analysis [14, 4] are the fields in which these models have shown greater applicability.

Colloquially, a CNN as presented by Yann LeCun and others, is a model that uses multiple layers of feature detectors that have local receptive fields and shared parameters interleaved with sub-sampling layers [15, 23, 24]. For attaining translation invariance, by design, these sub-sampling layers discard spatial information [9], which, when applied to the classification task, assist in amplifying the aspects of its input that are useful for discriminating and suppress irrelevant variations that are not.

Translation invariance, however helpful in attaining a model that has the same classification when applied to entities in different viewpoints, inevitably requires training on lots of redundant data. This redundancy is artificially introduced to force the optimization process to find solutions that can not distinguish between different viewpoints of the same entity. Additionally, disregarding spatial information produces models incapable of dealing with recognition tasks, such as facial identity recognition, that require knowledge of the precise spatial relationships between high-level parts, like a nose or a mouth.

1.1 Capsule Networks

To address these drawbacks, adaptations on CNNs have been proposed. This project focuses on improving an existing equivariant approach, introduced by Sara Sabour, Geoffrey E. Hinton and Nicholas Frosst, called Capsule Networks

*Supported by Calouste Gulbenkian Foundation

[22, 8]. In parallel with a neural network, a Capsule Network [22, 8] is, in essence, multiple levels of capsule layers which, in part, are composed of various capsules. A capsule is a group of artificial neurons which learns to recognise an implicitly defined entity, in the network's input, and outputs a tensor representation, the **pose**, which captures the properties of that entity relative to an implicitly defined canonical version and an activation probability. The activation probability was designed to express the presence of the entity the capsule represents in the network's input. In the high-level capsules this activation probability corresponds to the inter-class invariant discriminator used for classification. Moreover, every capsule's pose is **equivariant**, meaning that as the entity moves over the appearance manifold, the pose moves by a corresponding amount.

Every capsule layer is either calculated from multiple feature maps, when they correspond to the **primary capsule layer**, or by a series of transformations, followed by a mechanism called **routing**, applied to the outputs of the previous capsule layer.

Disentangling the internal representations in viewpoint invariant, presence probability and viewpoint equivariant, instantiation parameters may prove to be a more generalizable approach for representing knowledge than the conventional CNN view, that only strives for representational invariance. Early evidence of this has been revealed by experimenting on the effects of varying individual components of the individual capsules. The MNIST trained network presented in [22], without being explicitly designed to, learned to capture properties such as stroke, width and skew. In addition to this, as mentioned by Hinton and others [22], the same network, despite never being trained with digits, that were subject to affine transformations, was able to accurately classify them during test time, which seems to suggest greater generalization capacity when in comparison with conventional CNNs.

The Capsule Network model proposed by Sabour et al. [22] uses, layer wise, dynamic routing. Since dynamic routing is a sequential extremely computationally expensive procedure, if the network were to be scaled, in order to be suited to solve more challenging datasets, would incur in overly expensive costs in training and inference time. Furthermore, when applied to many capsules, the gradient flow through the dynamic routing computations is dampened. This inhibits learning, regardless of the computational resources used. Additionally, both dynamic routing as well as other routing procedures proposed, a priori, establish the way in which the routing relations are modeled, which limits the expressiveness of the underlying model.

In this work, we propose two distinct ways in which the routing procedure can be discriminatively learned. In parallel with [8], we employ routing to local receptive fields with parameter sharing, in order to reduce vanishing gradients, take advantage of the fact that correlated capsules tend to concentrate in local regions and reduce the number of model parameters.

2 Related Work

Capsules were first introduced in [9], whereas the logic of encoding instantiation parameters was established in a transforming autoencoder.

More recently, further work on capsules [22] garnered some attention achieving state-of-the-art performance on MNIST, with a shallow Capsule Network using an algorithm named Dynamic routing.

Shortly after, a new Expectation-Maximisation routing algorithm was proposed in [8], and capsule vectors were replaced by matrices to reduce the number of parameters and also introduced convolutional capsules. State-of-the-art performance was achieved on the smallNORB dataset using a relatively small Capsule Network.

An analysis of Dynamic routing algorithm as an optimization problem was presented in [25] as well as a discussion of possible ways to improve Capsule Networks.

3 Routing Procedure

Routing consists of a dot-product self-attention procedure that assigns, for each output capsule, a distribution of **compatibility probabilities** to the transformed previous layer's capsules, the **capsule votes**. These compatibility probabilities, after multiplied to corresponding capsule votes are combined and result in the output capsule's pose. Furthermore, the routing procedure also assigns the activation probability to the respective output capsule, usually based on the amount of agreement between the votes with higher compatibility probabilities.

This procedure provides the framework for a consensus mechanism in the multi-layer capsule hierarchy, which, for the higher levels of the input's domain, will solve the problem of assigning parts to wholes. Additionally, it can be applied to local receptive fields with shared transformation matrices. Figure 1 contains a diagram representing how routing is applied convolutionally in one dimension. The 2D and 3D convolutional routing is extrapolated in the same manner as the usual 2D and 3D convolution would.

The vote transformations are traditionally linear transformations. However, due to some stability issues that occur during training, equation 1, proposed by [25], will be used instead. The capsule vote from the ith input capsule to the

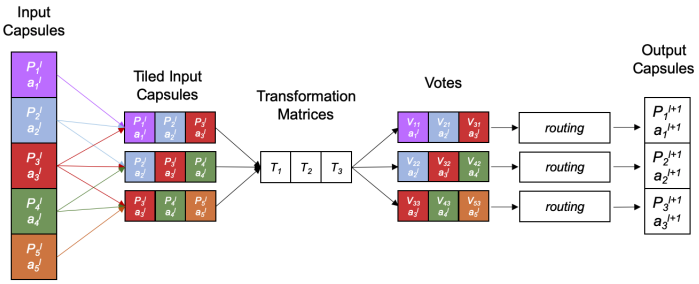

Figure 1: 1D Convolutional Routing(adapted from [6]). Respectively, P_i^l and a_i^l, denote the pose and activation corresponding to the ith capsule in the l capsule layer.

jth output capsule is obtained by multiplying the matrix W_{ij} to the pose of the ith input capsule P_i divided by the frobenius norm of W_{ij}.

$$V_{ij} = \frac{W_{ij}}{\|W_{ij}\|_{\mathcal{F}}} P_i \tag{1}$$

In algorithm 1, we present a generalization of the routing procedure that encompasses most of the routing algorithms proposed. The definition of the *activation* and *compatibility* functions is what caracterizes the particular routing procedure.

Algorithm 1 Generic Routing Mechanism

The routing algorithm returns the output capsule's pose μ and activation p given a subset of capsules γ. For some capsule $i \in \gamma$ the capsule activation is represented by a_i, the capsule vote is represented by V_i and the compatibility probability is represented by c_i. The symbol h represents some state values that might be shared either by the compatibility function across iterations or by the compatibility and activation functions.

procedure ROUTING(V, a, γ)
  $\forall i \in \gamma : c_i \leftarrow 1/\|\gamma\|$
  $h \leftarrow h_0$
  $\mu \leftarrow \frac{\sum_{i \in \gamma} c_i V_i}{\sum i \in \gamma c_i}$
  $p \leftarrow activation_\beta(\mu, h, V, c, \gamma)$
  for t iterations **do**
    $\forall i \in \gamma : c_i, h \leftarrow compatibility_\theta(c_i, h, V_i, a_i, \mu, p)$
    $\mu \leftarrow \frac{\sum_{i \in \gamma} c_i V_i}{\sum i \in \gamma c_i}$
    $p \leftarrow activation_\beta(\mu, h, V, c, \gamma)$
  end for
  return μ, p
end procedure

4 Learning the Routing Procedure

In hopes of improving the performance of Capsule Networks and of incorporating routing into the whole training process, instead of designing an alternative routing procedure, that necessarily constrains the parts to whole relations that can be modeled, we present methods for parameterizing it, such that, either for each layer or each network, the routing procedure itself can be discriminatively learned like any other model parameter.

In the following subsections we present two distinct alternatives. The first one exposes routing as a classic clustering algorithm based on the application of parametric kernel functions. The second takes a less structured approach that defines the activation and compatibility functions simply as neural networks.

4.1 Kernel Learning Approach

Since for each input, in every output capsule, the routing computations are reminiscent of an agglomerative fuzzy clustering algorithm, following the avenue taken in [25], in this subsection, routing it's analyzed as the optimization of a clustering-like objective function. The resulting cluster is interpreted as the agreement over the capsule votes and

is used as the routing procedure's output capsule's pose. Similarly with [25] we propose 2, inspired by the algorithm presented in [19].

$$\min \mathcal{L}(C, \mu) = -\sum_{i=1}^{n} c_i a_i \langle \mu, v_i \rangle_\theta + \lambda \sum_{i=1}^{n} c_i log c_i \tag{2}$$

subject to

$$\sum_{i=1}^{n} c_i = 1, c_i \in [0, 1], 1 \leq i \leq n \tag{3}$$

where $\langle \cdot, \cdot \rangle_\theta$ is a kernel function defined by θ, μ is the output capsule's pose, v_i and a_i are, respectively, the ith input capsule's vote and its activation, $C = [c_i]$ is a vector with $\mathbf{n}$ components, one for each input capsule's vote, c_i is the compatibility probability that represents the degree of similarity between the ith capsule's vote and the output capsule's pose and λ is a penalty factor for the negative entropy term applied to C.

The first term in the objective function represents the additive inverse of the weighted average of the similarity between the output pose and the ith input votes. The weights are the product of the input capsule's activation and the corresponding compatibility probability. In this way, the more activated and compatible an input capsule is the more significant its vote similarity is in the overall optimization process. Furthermore, the second term was chosen so that there is a penalty for the polarisation of the compatibility probabilities. This, in essence, was introduced so that the output capsule's pose is not only equal to one of the input votes but instead is an intricate mixture.

What ties this approach into the learning the routing procedure framework is that the parameter θ of the kernel function and the penalty factor λ are learned discriminatively using back-propagation during the training process.

The minimization problem is solved by partially optimizing $\mathcal{L}(C, \mu)$ for C and μ.

For a fixed C, μ is updated as

$$\mu = \sum_{i=1}^{n} c_i v_i \tag{4}$$

For a fixed μ, C is updated as follows: Using the Lagrangian multiplier technique, we obtain the unconstrained minimization problem 5.

$$\tilde{\mathcal{L}}(C, \alpha) = -\sum_{i=1}^{n} c_i a_i \langle \mu, v_i \rangle_\theta + \lambda \sum_{i=1}^{n} c_i log c_i + \alpha (\sum_{i=1}^{n} c_i - 1) \tag{5}$$

where α is the Lagrangian multiplier. If $(\hat{C}, \hat{\alpha})$ is a minimizer of $\tilde{\mathcal{L}}(C, \alpha)$ then the gradient must be zero. Thus,

$$\frac{\partial \tilde{\mathcal{L}}(\hat{C}, \hat{\alpha})}{\partial c_i} = -a_i \langle \mu, v_i \rangle_\theta + \lambda (1 + log \hat{c_i}) + \hat{\alpha} = 0, 1 \leq i \leq n \tag{6}$$

and

$$\frac{\partial \tilde{\mathcal{L}}(\hat{C}, \hat{\alpha})}{\partial \alpha} = \sum_{i=1}^{n} \hat{c_i} - 1 = 0 \tag{7}$$

From 6, we obtain

$$\hat{c_i} = exp(\frac{a_i \langle \mu, v_i \rangle_\theta}{\lambda}) exp(-1) exp(\frac{-\hat{\alpha}}{\lambda}), 1 \leq i \leq n \tag{8}$$

By substituting 8 into 7, we have

$$\sum_{i=1}^{n} \hat{c_i} = exp(-1) exp(\frac{-\hat{\alpha}}{\lambda}) \sum_{i=1}^{n} exp(\frac{a_i \langle \mu, v_i \rangle_\theta}{\lambda}) = 1 \tag{9}$$

which, when solved for $\hat{\alpha}$, results in

$$\hat{\alpha} = -\lambda log \frac{1}{exp(-1) \sum_{j=1} exp(\frac{a_j \langle \mu, v_j \rangle_\theta}{\lambda})} \tag{10}$$

It follows that by substituting 10 into 8, leading to

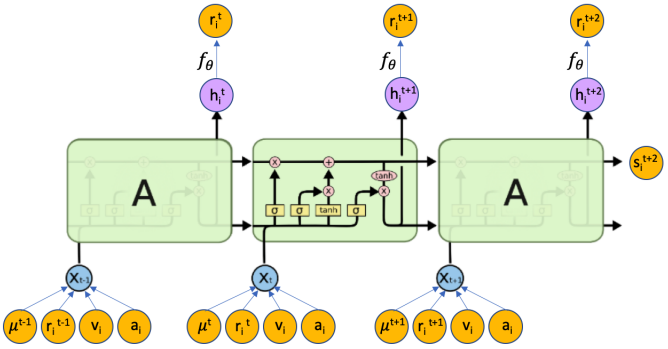

Figure 2: Diagram of of the LSTM cell used in the routing mechanism(adapted from [1]).

$$\hat{c}_i = \frac{exp(\frac{a_i \langle \mu, v_i \rangle_\theta}{\lambda})}{\sum_{j=1}^{n} exp(\frac{a_j \langle \mu, v_j \rangle_\theta}{\lambda})}, 1 \leq i \leq n \tag{11}$$

and C can be updated by 11.

Applied to the general routing procedure framework, presented in algorithm 1, we can take the *compatibility* function to be equation 12.

$$compatibility_\theta(r_i, h, V_i, a_i, \mu, p, \gamma) \doteq (\frac{exp(\frac{a_i \langle \mu, v_i \rangle_\theta}{\lambda})}{\sum_{j \in \gamma} exp(\frac{a_j \langle \mu, v_j \rangle_\theta}{\lambda})}, h) \tag{12}$$

In this way, we obtain a *compatibility* function that is parameterized by θ and λ and can be learned according to the specific machine learning problem in hand. The remaining function for the definition of the routing procedure is the *activation*, present in 13, which we defined to be a sigmoid applied to the linear transformation of the dot-product between the compatibility probabilities and the similarity between the corresponding votes and the final output capsule's pose.

$$activation_\beta(\mu, h, V, r, c, \gamma) \doteq sigmoid(\beta_1(\sum_{i \in \gamma} c_i \langle \mu, V_i \rangle_\theta) + \beta_2) \tag{13}$$

4.2 Connectionist Approach

An alternative approach to the more formal presented in the previous subsection, to modeling the routing procedure, is to allow the *activation* and *compatibility* functions in Algorithm 1 to be Neural Networks. More precisely we employed a LSTM cell [11], which is designed to keep track of arbitrary long-term dependencies, in conjunction with two distinct neural networks to obtain a learnable routing mechanism that plays an active role throughout the iterations.

Figure 2 presents a diagram of the LSTM cell employed. The input of the cell is a concatenation of the previous iteration intermediate output capsule pose μ^l, the capsule vote from the *ith* input capsule V_i and the corresponding routing intensity r_i and the activation a_i. The routing intensities are the unnormalized equivalent to the compatibility probabilities and are each initialized with the value one. After every single iteration step, the mentioned routing intensities go through a softmax function to obtain the corresponding compatibility probabilities. After initialized with zeros, the cell state s_i^t and hidden state h_i^t are updated throughout the iterations to obtain representations that are subsequently fed to two distinct neural networks.

The neural network f_θ is applied to the outputs of the hidden state to produce the routing intensities at the end of every iteration. Eventually, at the end of the iterative process, the cell states, correspondent to every input capsule, after combined, are fed to the neural network g_θ and result in the output capsule's final activation as indicated by 14.

$$p \doteq g_\theta(\sum_{i \in \gamma} c_i s_i) \tag{14}$$

The parameters from the LSTM and both neural networks are shared across every single input capsule. Additionally, the dimension of the LSTM's cell state and hidden state, as well as the architectures of the neural networks used, become hyperparameters.

Table 1: Capsule Network architecture

Layer	Details	Output shape
Input		$? \times 32 \times 32 \times 1$
Convolutional layer + relu + BatchNorm	K=5, S=2, Ch=64	$? \times 16 \times 16 \times 64$
Primary Capsules	K=1, S=1, Ch=8	$? \times 16 \times 16 \times 8 \times (4 \times 4 + 1)$
Convolutional Capsule Layer 1	K=3, S=2, Ch=16	$? \times 7 \times 7 \times 16 \times (4 \times 4 + 1)$
Convolutional Capsule Layer 2	K=3, S=1, Ch=16	$? \times 5 \times 5 \times 16 \times (4 \times 4 + 1)$
Capule Class Layer	flatten, O=5	$? \times 1 \times 1 \times 5 \times (4 \times 4 + 1)$

1. Specification of the Capsule Network model used in the experiment with SmallNORB dataset. **K** denotes convolutional kernel size, **S** stride, **Ch** number of output chanels and **O** number of classes.

Table 2: Connectionist approach routing parameters.

layer	hidden layers in f_θ	hidden layers in g_θ	number of units in the lstm's hidden and cell states
Convolutional Capsule Layer 1	[]	[]	16
Convolutional Capsule Layer 2	[32,32]	[64,64]	16
Capule Class Layer	[64,64]	[124,124]	16

2. The detailed specification of the hyperparameters used in the experiments with the Connectionist approach's routing mechanism. The representation of the hidden layers is a list where the *ith* list element corresponds to the number of neurons in the *ith* hidden layer.

5 Experiments

In order to evaluate the effectiveness of the proposed routing algorithms, when compared with **Expectation-Maximization**, introduced in [8], experiments were conducted in both MNIST dataset [17] and smallNORB [18]. They consist in training the same Capsule Network architecture, for 100 epochs, with every routing procedure running for three iterations, for both of the proposed algorithms and the one present in [8] and comparing the respective test set results. MNIST was selected, in the early stages, as proof of concept. smallNORB was chosen due to it being much closer to natural images and yet devoid of context and color.

All the experiments where made using a Tensorflow [2] implementation of the underlying models [1]. The optimizer used was Adam [13] with a learning rate of 3-e3 scheduled by an exponential decay. Additionally, the models were trained using kaggle notebooks, approximately 40 hours each, a free research tool for machine learning that provides a Tesla P100 Nvidia GPU as part of an accelerated computing environment.

Apart from the implementation of the routing procedure, all of the models contained in the experiments instantiate the Capsule Network architecture present in Table 1 and use 4x4 matrices as poses. The model is a slight modification of the smaller Capsule Network used in [8]. More precisely, the compatibility probabilities are not shared across the convolutional channel, after the initial convolutions batch normalization [12] is applied, and the transformation used is the one described in equation 1. Applied to smallNORB, the model has 86K parameters, excluding the ones pertaining to the different routing procedures.

The model present in table 1 contains three layers to which routing is applied. When used in the described experiments, the routing algorithm presented in section 4.2(**Connectionist**), used the hyperparameters described in table 2. Moreover, the routing algorithm presented in section 4.1(**Kernel Learning**) used the kernel defined in equation 15, a mixture of gaussian kernels. The first two layers in the Capsule Network have Q = 4 and the last one has Q = 10.

$$\langle x, x' \rangle_\theta \doteq \sum_{q=1}^{Q} \theta_1^q exp(-2\pi^2 \frac{\|x - x'\|^2}{(\theta_2^q)^2}) \tag{15}$$

The choice of routing hyperparameters was based on making the routing procedure in the deeper layers have more parameters. What motivated this design choice was the intuition that, the more complex the features, further complex needed to be the routing.

The results pertaining to all of the experiments are contained in table 3. The table contains the percentual error rate achieved in the evaluation of the models with the test set of the corresponding datasets.

[1] https://github.com/Goncalo-Faria/learning-inner-consensus

Table 3: Test set error rate.

Routing	MNIST	smallNORB
Connectionist	0.53%	6.7%
Expectation-Maximization	0.90%	4.6%
Kernel Learning	1.09%	8.25%

6 Conclusion

Building on the work of [8] and [22], we propose two distinct ways in which Capsule Network's inner routing mechanism can be learned with backpropagation.

Compared with Expectation-Maximization and the kernel learning approach, the routing procedure derived with the connectionist approach achieves greater accuracy on the MNIST dataset. However, the more sophisticated smallNORB dataset appears to be a more challenging task for the proposed routing procedures, since the experimental results suggest that, for this Capsule Network architecture, Expectation-Maximization is a superior algorithm.

ACKNOWLEDGMENTS A special thank you to the Gulbenkian Foundation, the scientific committee of the Artificial Intelligence Program and this project's tutor Professor Cesar Analide.

References

[1] Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2019-08-15.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A Neural Probabilistic Language Model. In *Journal of Machine Learning Research*, 2003.

[4] Q. Dou, H. Chen, L. Yu, L. Zhao, J. Qin, D. Wang, V. C. Mok, L. Shi, and P. A. Heng. Automatic Detection of Cerebral Microbleeds from MR Images via 3D Convolutional Neural Networks. *IEEE Transactions on Medical Imaging*, 2016.

[5] K. L. Du and M. N. Swamy. *Neural networks in a softcomputing framework*. 2006.

[6] A. D. Gritzman. 2019.

[7] S. Haykin. *Neural networks: a comprehensive foundation*. 1994.

[8] G. Hinton, S. Sabour, and N. Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[9] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.

[10] B. Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., ... Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 2012.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] J. Kleesiek, G. Urban, A. Hubert, D. Schwarz, K. Maier-Hein, M. Bendszus, and A. Biller. Deep MRI brain extraction: A 3D convolutional neural network for skull stripping. *NeuroImage*, 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *ImageNet Classification with Deep Convolutional Neural Networks*, 2012.

[16] Y. LeCun. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, 1989.

[17] Y. LeCun and C. Cortes. The mnist database of handwritten digits. 2005.

[18] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, 2:II–104 Vol.2, 2004.

[19] M. J. Li, M. K. Ng, Y. M. Cheung, and J. Z. Huang. Agglomerative fuzzy K-Means clustering algorithm with selection of number of clusters. *IEEE Transactions on Knowledge and Data Engineering*, 2008.

[20] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon. A proposal for the Dartmouth summer research project on artificial intelligence. 2012.

[21] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[22] N. F. Sara Sabour and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, 2017.

[23] P. Sermanet and Y. Lecun. Traffic sign recognition with multi-scale convolutional networks. In *Proceedings of the International Joint Conference on Neural Networks*, 2011.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[25] D. Wang and Q. Lui. An Optimization View on Dynamic Routing Between Capsules. *Journal of Geotechnical and Geoenvironmental Engineering*, 2018.

[26] Z. C. Q. V. L. M. N. W. M. M. K. Y. C. Q. G. K. M. Yonghui Wu, Mike Schuster. 2016.

[27] Yoshua Bengio. Deep Learning Book. *MIT Press*, 2015.