Journal of Information Hiding and Multimedia Signal Processing ©2023 ISSN 2073-4212
Ubiquitous International Volume **14**, Number **3**, September **2023**

Operator-N Layer Construction for Optimizing Capsule Network Methods in Image Classification Problems

Ridho Nur Rohman Wijaya[1], Budi Setiyono[2], Mahmud Yunus[3], Dwi Ratna Sulistyaningrum[4]

Departement of Mathematics
Institut Teknologi Sepuluh Nopember Surabaya, Indonesia
[1]ridhonurrohmanwijaya@gmail.com, [2]masbudisetiyono@gmail.com, [3]yunusm@matematika.its.ac.id, [4]dwratna@gmail.com

Correspond Author : masbudisetiyono@gmail.com
Received June 2023; revised August 2023

Abstract. *Capsule Network (CapsNet) is an image classification method that has successfully demonstrated excellent performance in Deep Learning. The main concept of CapsNet is to change image features into capsules and create entities that better represent information from these features. However, capsule formation causes an increase in the number of training parameters and makes the computation time longer. Moreover, the CapsNet method is less effective in processing images with complex backgrounds because it has limitations in feature extraction. We propose a mathematical approach by introducing an operator-N layer to improve CapsNet performance. The operator-N is an operator constructed using Euclidean norms, which functions to shrink dimensions and speed up the process of translation equivariance from a capsule. According to experimental results on MNIST, Fashion MNIST, and Kuzushiji MNIST datasets, the proposed operator-N layer significantly improves accuracy up to 1.72% with 2.91 times faster computation time than the original CapsNet. Furthermore, the total parameters used during the training process have been reduced to 15.7%. Our research provides valuable insights for overcoming the challenges posed by feature extraction limitations and paves the way for more efficient image classification methods in the future.*
Keywords: Capsule Network, Deep Learning, Feature Extraction, Norm, Operator.

1. **Introduction.** Deep Learning has been growing in recent years. Deep learning has become an invaluable tool for various applications in our daily lives, including medical image processing [1], sentiment analysis [2], voice recognition [3], and others. Deep learning popularity began with the Convolutional Neural Networks (CNN) method to solve the image classification problem [4]. At its inception, CNN architecture is designed with a simple network, starting with a few layers and growing to thousands of layers. This development gave birth to new methods such as VGGNet [5], GoogleNet [6], ResNet [7], and CapsNet [8]. The Capsule Network (CapsNet) method is a new method whose main motivation is removing the pooling layer and adding a component known as a capsule. Based on the statement of Sabour *et al* [8], the pooling technique retrieves translation equivariance with a rough working mechanism. An example is Max pooling which only picks neurons with the highest activation, not those with the most relevant information to the problem at hand. The pooling process can reduce information from

the image so that it cannot recognize the spatial relations between specific components of the objects in the image. Therefore, the CapsNet method emerged with a better translation equivariance mechanism, which is "dynamic routing". Based on this routing process, the CapsNet method is produced which improves CNN performance to recognize spatial relations such as slope, position, and thickness in images. CapsNet performance results have been proven on [8], and it has also been proven that for a limited amount of training data, the CapsNet method is better than CNN in terms of [9] accuracy. One of the problems with the CapsNet method is the large and long computation [10]. CapsNet requires significant computational resources to obtain more optimal results. Hence, some studies that focus on efficient computing time with optimal accuracy, such as [11] which accelerates dynamic routing by up to three times. There is also development that focuses on the efficiency of the routing process, several routing alternatives that can be used to make the CapsNet method even better are EMCaps [12], OptimCaps [13], GroupCaps [14], and AttnCaps [15].

An interesting concept in the routing process is using Euclidean norms that do not exist in the CNN method. The utilization of norms in the routing process is facilitated by the fundamental concept of capsules in CapsNet. This capsule is a vector consisting of a group of neurons representing several parameters and the length of the vector indicates the possibility of a specific entity [16]. Therefore the capsule can store important information such as position, tilt, and thickness. However, we realize that making capsules in the CapsNet method is very simple: taking each neuron in a different layer and turning it into a vector. Even though the information on the neurons in each layer is not necessarily needed in every condition, a better method of retrieving neurons is needed so to form capsules that better represent the specific entities that exist. Hence, our proposed approach involves incorporating the operator-N, constructed from Euclidean norms, to effectively reduce image dimensions. This operator-N plays a crucial role in shrinking dimensions and accelerating translation equivariance at each layer, leading to the generation of capsules that better capture the relevant information. By introducing this dimension reduction technique as an additional layer in the CapsNet method, we achieve a modified CapsNet model that offers enhanced accuracy and faster computation capabilities.

2. **The Proposed Method.**

This work is to optimize the performance of CapsNet method by proposing a new layer using operator-N operation, which is constructed using a norm. First, we define the input and output in this method. Let D denote a dataset consisting of pairs of images and labels, $D = \{(\mathbf{X}_i, y_i) \mid \mathbf{X}_i \in \mathbb{R}^{r\times c\times d}, y_i \in \mathbb{R}, i = \underline{n} = 1, 2, \ldots, n\}$, then we get set of $\mathbf{X} = \{\mathbf{X}_i \mid \mathbf{X}_i \in \mathbb{R}^{r\times c\times d}, i = \underline{n}\}$ and $\mathbf{Y} = \{y_i \mid y_i \in \mathbb{R}, i = \underline{n}\}$ as the CapsNet model inputs (CN). Therefore, the output is a label prediction $\hat{y} = CN(\hat{\mathbf{X}}) \in \mathbb{R}$ when the model CN is used to predict test data $\hat{\mathbf{X}} \in \mathbb{R}^{r\times c\times d}$. The next step for proposed CapsNet was formed with the architecture shown in FIGURE 1.

2.1. **Proposed Capsule Network.**

2.1.1. *The architecture.* The modified CapsNet architecture consists of a convolutional layer, primary capsule layer, operator-N layer, digital capsule layer, and fully connected layer as reconstruction regularization. To simplify it, we denoted a convolution layer and a dense layer with functions f_{conv} and f_{dense} without explaining their process. Hence, the result in convolutional layer for $\mathbf{X}$ as the input is $\mathbf{X}_{(1)} = \{\mathbf{X}_i^{(1)} \mid \mathbf{X}_i^{(1)} \in \mathbb{R}^{r_1\times c_1\times d_1}, i = \underline{n}\}$

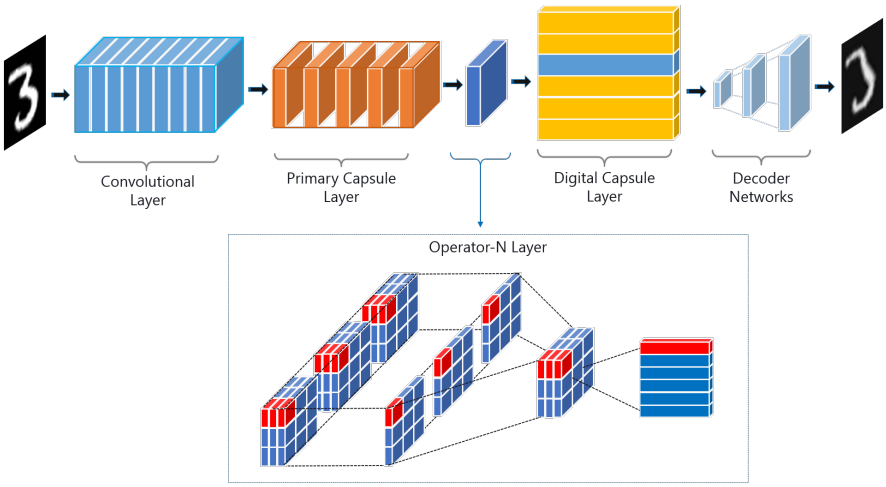

FIGURE 1. The architecture of the proposed CapsNet with Operator-N Layer.

with

$$\mathbf{X}_i^{(1)} = f_{conv}(\mathbf{X}_i, d_{k_{(1)}}, (k_{(1)}, k_{(1)}), (s_{(1)}, s_{(1)})), \ \forall \mathbf{X}_i \in \mathbf{X}, \tag{1}$$

for $(k_{(1)}, k_{(1)})$ and $(s_{(1)}, s_{(1)})$ is kernel size of $d_{k_{(1)}}$ kernels and stride at the convolutional layer. Furthermore, we get $\mathbf{X}_{(2)} = \{\mathbf{X}_i^{(2)} \mid \mathbf{X}_i^{(2)} \in \mathbb{R}^{r_2 \times c_2 \times d_2}, i = \underline{n}\}$ as result in primary capsule layer with $d_{k_{(2)}}$ kernels of $(k_{(2)}, k_{(2)})$ and stride $(s_{(2)}, s_{(2)})$,

$$\mathbf{X}_i^{(2)} = f_{conv}(\mathbf{X}_i^{(1)}, d_{k_{(2)}}, (k_{(2)}, k_{(2)}), (s_{(2)}, s_{(2)})), \forall \mathbf{X}_i^{(1)} \in \mathbf{X}_{(1)}. \tag{2}$$

For the next stage, $\mathbf{X}_{(2)}$ as input at the operator-N layer and has an output $\mathbf{X}_{(3)} = \{\mathbf{X}_i^{(3)} \mid \mathbf{X}_i^{(3)} \in \mathbb{R}^{r_3 \times c_3 \times d_3}, i = \underline{n}\}$, the complete process will be explained in Proposed Operator-N Layer section.

The next step is to create the digital capsule layer. At this point, the information from previous stage, $\mathbf{X}_{(3)}$, is turned into a collection of vectors that hold specific data, which is commonly referred to as capsules in this method. The extraction of this information employs a routing mechanism known as Dynamic Routing. The routing process is a translation equivariance mechanism from the collection of capsules in a specific layer to a subsequent layer. Therefore, before the routing process is carried out, the value of $\mathbf{X}_{(3)}$ must be reformed into a set of capsule collections, i.e. $\mathbf{V}_{(1)} = \{\mathbf{V}_i^{(1)} \mid i = \underline{n}\}$ with $\mathbf{V}_i^{(1)} = \{\mathbf{u}_{i,k} \mid \mathbf{u}_{i,k} \in \mathbb{R}^{n_1}, k = \underline{m}_1\}$ such that it can be the input of the routing process. In this case, , the variable n_1 denotes the length of the capsule vector, while the value of $\mathbf{V}_i^{(1)}$ can be determined using the algorithm provided in Algorithm 1, ensuring that

$$\mathbf{V}_i^{(1)} = f_{vec}(\mathbf{X}_i^{(3)}, n_1). \tag{3}$$

2.1.2. *Routing Process.* The routing process is a translation equivariance mechanism from the collection of capsules in a specific layer to a subsequent layer. In this case, the retrieved information spans from $\mathbf{V}_{(1)}$ to a set of capsule collections $\mathbf{V}_{(2)} = \{\mathbf{V}_i^{(2)} \mid i = \underline{n}\}$ with $\mathbf{V}_i^{(2)} = \{\mathbf{v}_{i,\kappa} \mid \mathbf{v}_{i,\kappa} \in \mathbb{R}^{n_2}, k = \underline{m}_2\}$. There exists a weight matrix from capsule k to capsule κ denoted as $\mathbf{W}_{k\kappa} \in \mathbb{R}^{n_2 \times n_1}$ such that for every i image, we get the prediction vector $\hat{\mathbf{u}}_{\kappa|k} \in \mathbb{R}^{n_2}$ follows the following equation:

$$\hat{\mathbf{u}}_{\kappa|k} = \mathbf{W}_{k\kappa}\mathbf{u}_{i,k}, \tag{4}$$

Algorithm 1 Vector Building

Input: The input image $\mathbf{X}_i^{(3)} \in \mathbb{R}^{r_3 \times c_3 \times d_3}$, vector length n_1
Output: The set of capsules $\mathbf{V}_i^{(1)}$

procedure $f_{vec}(\mathbf{X}_i^{(3)}, n_1)$
  $\mathbf{V}_i^{(1)} \leftarrow \emptyset$
  for $k = 1 \rightarrow d_3/n_1$ **do**
    $\mathrm{k} = n_1(k-1)+1$
    for $l = 1$ **to** $r_3, j = 1$ **to** c_3 **do**
      $\mathbf{u}_{i,k} \leftarrow \mathbf{X}[l, j, \mathrm{k} : \mathrm{k} + n_1]$, $\mathbf{V}_i^{(1)} \leftarrow \mathbf{V}_i^{(1)} \bigcup \{\mathbf{u}_{i,k}\}$
    end for
  end for
  return $\mathbf{V}_i^{(1)}$
end procedure

hence, we get the vector $\mathbf{s}_\kappa \in \mathbb{R}^{n_2}$ as the weighted sum of the vectors $\hat{\mathbf{u}}_{\kappa|k}$, which are

$$\mathbf{s}_\kappa = \sum_{k=1}^{m_1} c_{k\kappa} \hat{\mathbf{u}}_{\kappa|k}, \tag{5}$$

where $c_{k\kappa} \in \mathbb{R}$ is the coupling coefficient obtained from dynamic routing process for weight $w_{k\kappa} \in \mathbb{R}$, in other words for each capsule k and capsule j are obtained

$$c_{k\kappa} = \frac{e^{w_{k\kappa}}}{\sum_{l=1}^{m_2} e^{w_{k\kappa}}}. \tag{6}$$

Based on this explanation, the vector value for the i-th image on capsule κ can be obtained by applying the squashing function $f_{squash} : \mathbb{R}^{n_2} \rightarrow \mathbb{R}^{n_2}$ with

$$\mathbf{v}_{i,\kappa} = f_{squash}(\mathbf{s}_\kappa) = \frac{\|\mathbf{s}_\kappa\|^2}{1 + \|\mathbf{s}_\kappa\|^2} \cdot \frac{\mathbf{s}_\kappa}{\|\mathbf{s}_\kappa\|}. \tag{7}$$

The routing process for each element in $\mathbf{V}(1)$ to $\mathbf{V}(2)$ is further demonstrated in Algorithm 2.

2.1.3. *Reconstruction Regularization.* The final outcome of the routing process is a set of capsules that encapsulate the characteristic values of the given input image. At this point, we can calculate loss value of the training results. However, to enhance the classification accuracy, the CapsNet method incorporates a fully connected layer that reconstructs capsule collection based on the routing outputs. This reconstruction process is a regularization technique to assess performance of the prior stage's routing outputs. The reconstruction is accomplished through three dense layers, each containing a specific number of neurons $\tilde{n}_1$, $\tilde{n}_2$, and $\tilde{n}_3$ with $\tilde{n}_3 = r \times c \times d$. For example, if we denote the reconstructed outputs at $\tilde{n}_1$, $\tilde{n}_2$, and $\tilde{n}_3$ as $\tilde{\mathbf{X}}_{(1)} = \{\tilde{\mathbf{X}}_i^{(1)} \mid i = \underline{n}\}$, $\tilde{\mathbf{X}}_{(2)} = \{\tilde{\mathbf{X}}_i^{(2)} \mid i = \underline{n}\}$, and $\tilde{\mathbf{X}}_{(3)} = \{\tilde{\mathbf{X}}_i^{(3)} \mid i = \underline{n}\}$, then

$$\tilde{\mathbf{X}}_i^{(1)} = f_{dense}(\mathbf{V}_i^{(2)}, \tilde{n}_1),\ \tilde{\mathbf{X}}_i^{(2)} = f_{dense}(\tilde{\mathbf{X}}_i^{(1)}, \tilde{n}_2),\ \tilde{\mathbf{X}}_i^{(3)} = f_{dense}(\tilde{\mathbf{X}}_i^{(2)}, \tilde{n}_3). \tag{8}$$

Based on this, we calculate two losses: the margin loss from the routing process and the reconstruction loss from the reconstruction regularization process.

Algorithm 2 Routing Mechanism

Input: The set of capsules $\mathbf{V}_i^{(1)}$, r routing
Output: The set of capsules $\mathbf{V}_i^{(2)}$

procedure ROUTING($\mathbf{V}_i^{(1)}, r_n$)
  $\mathbf{V}_i^{(2)} \leftarrow \emptyset$ ▷ initialization
  for $i = 1$ **to** n **do**
    for $k = 1$ **to** $m_1, \kappa = 1$ **to** m_2 **do**
      $\mathbf{W}_{k\kappa} \leftarrow Random(n_2, n_1)$, $w_{k\kappa} \leftarrow 0$, $\hat{\mathbf{u}}_{\kappa|k} \leftarrow \mathbf{W}_{k\kappa}\mathbf{u}_{i,k}$
    end for
    for $r = 1$ **to** r **do** ▷ routing process
      for $k = 1$ **to** $m_1, \kappa = 1$ **to** m_2 **do**
        $c_{k\kappa} \leftarrow e^{w_{k\kappa}} / \sum_{l=1}^{m_2} e^{w_{k\kappa}}$
      end for
      for $\kappa = 1$ **to** m_2 **do**
        $\mathbf{s}_\kappa \leftarrow \sum_{k=1}^{m_1} c_{k\kappa}\hat{\mathbf{u}}_{\kappa|k}$, $\mathbf{v}_{i,\kappa} \leftarrow f_{squash}(\mathbf{s}_\kappa)$
      end for
      for $k = 1$ **to** $m_1, \kappa = 1$ **to** m_2 **do**
        $w_{k\kappa} \leftarrow w_{k\kappa} + \hat{\mathbf{u}}_{\kappa|k} \cdot \mathbf{v}_{i,\kappa}$
      end for
    end for
    $\mathbf{V}_i^{(2)} \leftarrow \mathbf{V}_i^{(2)} \bigcup \{\mathbf{v}_{i,\kappa}\}$
  end for
  return $\mathbf{V}_i^{(2)}$
end procedure

2.1.4. *Loss Function.* The value of the loss $\mathcal{L}_i$ for the i-th image is calculated for each capsule in the DigitCaps layer. As mentioned earlier, there are two loss components in the CapsNet method: the margin loss $\mathcal{M}_i$ and the reconstruction loss $\mathcal{R}_i$. The margin loss $\mathcal{M}_i = \sum_{\kappa=1}^{m_2} M_\kappa$ represents the cumulative loss of each capsule κ, and it is calculated using the equation:

$$M_\kappa = \mathcal{T}_\kappa \left(\max\left\{0, v^+ - \|\mathbf{v}_{i,\kappa}\|\right\}\right)^2 + \lambda(1 - \mathcal{T}_\kappa)\left(\max\left\{0, \|\mathbf{v}_{i,\kappa}\| - v^-\right\}\right)^2, \tag{9}$$

where $\mathcal{T}_k$ is 1 if capsule κ is the label of the actual target and 0 otherwise, $v^+ = 0.9$, $v^- = 0.1$, and λ parameters reduce the effect of loss on labels that are not actual targets. The calculation of the reconstruction loss value involves using the mean square error, which is as follows: $\mathcal{R}_i = \text{MSE}(\mathbf{X}_i - \tilde{\mathbf{X}}_i^{(3)})$. Hence, the loss value is obtained.

$$\mathcal{L}_i = \mathcal{M}_i + \alpha\mathcal{R}_i, \tag{10}$$

where α is a parameter that determines the relative impact of the reconstruction loss.

2.2. **Proposed Operator-N Layer.** The operator-N is a mapping operator that transforms a finite-dimensional vector into a real value using operations based on Euclidean norms. Euclidean norms are chosen for this purpose because they are widely used and there exists a norm equivalence theorem, which states that all norms are equivalent. This means that the choice of norm does not significantly affect the magnitude of the result.

2.2.1. *Operator-N.* We constructed multiple operator-Ns as variations of operators built using Euclidean norms.

$$\|\mathbf{x}\| = \left(\sum_{k=1}^{n} |x_k|^2\right)^{1/2}, \ \mathbf{x} = (x_1, x_2, \ldots, x_p) \in \mathbb{R}^p. \tag{11}$$

Three variations were constructed, denoted as $\mathtt{N}_1 : \mathbb{R}^p \to [0, 1)$, $\mathtt{N}_2 : \mathbb{R}^p \to (0, 1]$, and $\mathtt{N}_3 : \mathbb{R}^p \to [\frac{1}{2}, 1)$, which is as follows:

$$\mathtt{N}_1 = \frac{\|\mathbf{x}\|}{1 + \|\mathbf{x}\|}, \tag{12}$$

$$\mathtt{N}_2 = e^{-\frac{\|\mathbf{x}\|^2}{2}}, \tag{13}$$

$$\mathtt{N}_3 = \frac{1}{1 + e^{-\|\mathbf{x}\|}}. \tag{14}$$

The formation of the three norms refers to the normalization, exponential, and sigmoid functions. As explained earlier, the operator-N layer is employed to obtain $\mathbf{X}_{(3)}$ from the input $\mathbf{X}_{(2)}$. This process is carried out by taking the image value at a certain depth and transforming it into a new real value using the operator-N. We call this term dimension shrinking, which means we shrink $\mathtt{s}$ dimensions to get better information. This process involves taking the image value at a specific depth and transforming it into a new real value using the operator-N. We refer to this process as "dimension shrinking," as it aims to reduce $\mathtt{s}$ dimensions of the vectors to capture more informative features. In theory, the CapsNet method imitates the functioning of the human brain by utilizing capsules, which are vectors capable of representing spatial relations in images, such as position, slope, thickness, and more. Incorporating an operator-N layer can enhance and accelerate the extraction of spatial relation information compared to the absence of this layer. More details are shown in Algorithm 3.

2.2.2. *Theoretical Analysis.* We conduct a theoretical analysis to establish the existence and well-defined nature of the operator-N. First, we show the theory of norm equivalence.

Theorem 2.1. [17] *Every two norms in a finite-dimensional vector space are equivalent.*

Algorithm 3 Operator-N Layer

Input: The input image $\mathbf{X}_i^{(2)} \in \mathbb{R}^{r_2 \times c_2 \times d_2}$, shrink size $\mathtt{s}$, operator $\mathtt{N}$
Output: The output image $\mathbf{X}_i^{(3)}$

procedure $f_{opN}(\mathbf{X}_i^{(2)}, \mathtt{s}, \mathtt{N})$
  $\mathbf{X}_i^{(3)} \leftarrow \mathbf{0}$
  for $k = 1 \to d_2/\mathtt{s}$ **do**
    $\mathrm{k} = \mathtt{s}(k-1) + 1$
    for $l = 1$ **to** $r_2, j = 1$ **to** c_2 **do**
      $\mathbf{x} \leftarrow \mathbf{X}[l, j, \mathrm{k} : \mathrm{k} + \mathtt{s}]$, $\mathbf{X}_i^{(3)}[l, j, k] \leftarrow \mathtt{N}(\mathbf{x})$
    end for
  end for
  return $\mathbf{X}_i^{(3)}$
end procedure

We know that any norms ϕ and ψ in a vector space with finite dimensions $\mathbb{V}$ are said to be equivalent if there are constants $A > 0$ and $B > 0$ such that $A\phi(\mathbf{v}) \leq \psi(\mathbf{v}) \leq B\phi(\mathbf{v})$ for each $\mathbf{v} \in \mathbb{V}$. Hence, we have the flexibility to select any norm as the basis for constructing the operator-N. In our research, we choose the Euclidean norm. It's important to mention that the Euclidean norm is already an established and well-defined concept in mathematics. As a result, the operators constructed from this norm will also be well defined. Consequently, we can confidently assert that the operator-N is indeed well defined. Considering that Euclidean norms are continuous, it follows that dividing two continuous functions also yields a continuous function. Additionally, the exponential function is known to be continuous. Hence, we can conclude that the operators $\mathtt{N}_1(\cdot)$, $\mathtt{N}_2(\cdot)$, and $\mathtt{N}_3(\cdot)$ are all continuous operators. Next, we determine the Supremum and Infimum of each operator. Let $\mathbf{x} = (x_1, x_2, \ldots, x_p) \in \mathbb{R}^p$, then

$$\begin{aligned} \inf_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_1(\mathbf{x}) &= \inf_{\mathbf{x}\in\mathbb{R}^n} \frac{\|\mathbf{x}\|_2}{1+\|\mathbf{x}\|_2}, & \sup_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_1(\mathbf{x}) &= \sup_{\mathbf{x}\in\mathbb{R}^n} \frac{\|\mathbf{x}\|_2}{1+\|\mathbf{x}\|_2} \\ &= 0 = \min_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_1(\mathbf{x}) & &= 1 \neq \max_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_1(\mathbf{x}) \end{aligned} \tag{15}$$

$$\begin{aligned} \inf_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_3(\mathbf{x}) &= \inf_{\mathbf{x}\in\mathbb{R}^n} \exp\left(-\frac{\|\mathbf{x}\|_2^2}{2}\right), & \sup_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_3(\mathbf{x}) &= \sup_{\mathbf{x}\in\mathbb{R}^n} \exp\left(-\frac{\|\mathbf{x}\|_2^2}{2}\right) \\ &= 0 \neq \min_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_3(\mathbf{x}) & &= 1 = \max_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_3(\mathbf{x}) \end{aligned} \tag{16}$$

$$\begin{aligned} \inf_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_4(\mathbf{x}) &= \inf_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{1+\exp(-\|\mathbf{x}\|_2)}, & \sup_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_4(\mathbf{x}) &= \sup_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{1+\exp(-\|\mathbf{x}\|_2)} \\ &= \frac{1}{2} = \min_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_4(\mathbf{x}) & &= 1 \neq \max_{\mathbf{x}\in\mathbb{R}^n} \mathtt{N}_4(\mathbf{x}) \end{aligned} \tag{17}$$

Thus, we can conclude that $\mathtt{N}_1 : \mathbb{R}^p \to [0, 1)$, $\mathtt{N}_2 : \mathbb{R}^p \to (0, 1]$, and $\mathtt{N}_3 : \mathbb{R}^p \to [\frac{1}{2}, 1)$. The performance of operator-N is evaluated based on the achieved accuracy and the computation time. Suppose the modified CapsNet method demonstrates improved accuracy with reduced computation time. In that case, the inclusion of operator-N has effectively enhanced the performance of the original CapsNet method. Mathematically, the performance of a computation can be quantified by the number of parameters.

2.3. **Parameter Analysis.** The performance of an algorithm can be measured by calculating its computational complexity or the number of operations. External factors such as memory capacity and GPU usage also play a role. However, in the context of deep learning methods, calculating the complexity using Big O notation is often unnecessary. Instead, we can evaluate the performance based on the total number of parameters used in the model. Let $\mathtt{P}(\mathbf{A}, \mathbf{B})$ represents the total parameters required to pass neuron information from $\mathbf{A}$ to $\mathbf{B}$, then we get the theorem as follows:

Theorem 2.2. *The total number of parameters used in the modified CapsNet method, including the operator-N, is lower than that of the original method.*

Proof. We take $\mathbf{X}_i^{(2)} \in \mathbb{R}^{r_2\times c_2\times d_2}$ and $\mathbf{X}_i^{(3)} \in \mathbb{R}^{r_3\times c_3\times d_3}$ for comparison, the original CapsNet calculates total parameter as $\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{V}_i^{(2)})$ and our CapsNet as $\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{X}_i^{(3)}) + \mathtt{P}(\mathbf{X}_i^{(3)}, \mathbf{V}_i^{(2)})$, Our goal is to demonstrate that the sum of all parameters $\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{X}_i^{(3)}) + \mathtt{P}(\mathbf{X}_i^{(3)}, \mathbf{V}_i^{(2)}) < \mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{V}_i^{(2)})$. Note that $\mathbf{X}_i^{(3)}$ is the convolution result from $\mathbf{X}_i^{(2)}$, then $r_2 \geq r_3$, $c_2 \geq c_3$, and $d_2 > d_3$. Accordingly, a set of weight matrices is formed to transform vectors of size $n_1 \times 1$ from $\mathbf{X}_i^{(2)}$, which has a total of $r_2 \times c_2 \times d_2/n_1$ capsules, to a set of vectors of size $n_2 \times 1$ in $\mathbf{V}_i^{(2)}$, which has m_2 capsules. Therefore, the

total parameter $\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{V}_i^{(2)}) = r_2 \times c_2 \times d_2 \times n_2 \times m_2$. Similarly, we have $\mathtt{P}(\mathbf{X}_i^{(3)}, \mathbf{V}_i^{(2)}) = r_3 \times c_3 \times d_3 \times n_2 \times m_2$. Mention that the process from $\mathbf{X}_i^{(2)}$ to $\mathbf{X}_i^{(3)}$ does not require any additional independent variables to use, As a result, we obtain $\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{X}_i^{(3)}) = 0$. Hence, it can be proven that

$$\mathtt{P}(\mathbf{X}_i^{(2)}, \mathbf{X}_i^{(3)}) + \mathtt{P}(\mathbf{X}_i^{(3)}, \mathbf{V}_i^{(2)}) = 0 + r_3 \times c_3 \times d_3 \times n_2 \times m_2 < r_2 \times c_2 \times d_2 \times n_2 \times m_2$$

The modified approach utilizing operator-N layers boasts a reduced total parameter count compared to the original method. □

3. **Experiments.**

3.1. **Datasets and System Setup.**

To evaluate the performance of our method, we achieved experiments using three benchmark datasets: MNIST [18], Fashion MNIST [19], and Kuzushiji-MNIST [20]. All datasets contain 70K grayscale images with size (28×28), 60K images are training data and 10K images are test data which have been divided into ten classes. We aim to evaluate the performance of the modified CapsNet method, referred to as "Our-CapsNet," by incorporating an operator-N layer in comparison to the original method, which we refer to as "Original-CapsNet". Therefore we use the same architecture for Original-CapsNet: two convolution layers, one digital capsule layer (DigitCaps), and a reconstruction layer. The first convolution layer has 256 convolution kernels with size (9×9), stride of 1, no padding, and the ReLU activation function. The second convolution layer, or primary capsule layer has 256 convolution kernels with size (9×9), a stride of 2, no padding, and ReLU activation function. This layer can be reshaped into a capsule layer which has 32 capsules with a size of 8 dimensions. The DigitCaps layer has ten capsules with 16 dimensions representing each class. The reconstruction layer contains three dense layers with neuron sizes of 512, 1024, and 784, respectively.

Our method shares a similar architecture with Original-CapsNet but includes additional components before the DigitCaps step. One of the key additions is the operator-N layer, which uses different depreciation operators and hyperparameters. We conducted a preliminary experiment to determine the optimal hyperparameters and operators. This preliminary experiment involved 16 training batches, the Adam optimizer function, and ten epochs. The training process was repeated three times, and the average accuracy from three separate tests was taken as the final result. Subsequently, we proceeded with 100 training epochs using the best hyperparameters identified in the preliminary experiment. Our method was implemented using TensorFlow and Python, with Tesla T4 GPU.

3.2. **Experimental Results.**

Table 1. Comparison of results for preliminary experiments on MNIST Dataset. (M:millions, s/E:seconds/Epoch)

Benchmarks	**Original-CapsNet**	**Our-CapsNet**		
		s		
		8	16	32
Params (M)	8,21	6.92	6.83	**6.78**
Time (s/E)	464	170	151	**128**
Accuracy	97.79%	**98.55%**	98.38%	98.47%

TABLE 2. Comparison of performance improvement for preliminary experiments on Fashion MNIST and Kuzushiji MNIST Dataset.

Dataset	**Our-CapsNet**		
	s		
	8	16	32
Fashion MNIST	**(+)1.70%**	(+)1.34%	(+)1.31%
Kuzushiji MNIST	**(+)2.92%**	(+)1.85%	(+)1.72%

TABLE 3. Comparison of test accuracy after applying different operators.

Dataset	**Our-CapsNet**			
	$\|\cdot\|$	$\mathtt{N}_1(\cdot)$	$\mathtt{N}_2(\cdot)$	$\mathtt{N}_3(\cdot)$
MNIST	98.60%	**98.79%**	97.70%	95.20%
Fashion MNIST	88.36%	**89.05%**	88.54%	85.12%
Kuzushiji MNIST	**92.78%**	91.31%	91.13%	87.66%

3.2.1. *Shrinkage Hyperparameter Selection.* We use three distinct hyperparameters for the $\mathtt{s}$ dimension shrinking at the operator-N layer step. In this experiment, we sampled 8, 16, and 32 dimensions. The corresponding results are presented in TABLE 1. Based on the results in TABLE 1, it is evident that shrinking the dimensions for all hyperparameters can improve the performance of the Original-CapsNet. The total parameters used decreased by 15.70%-17.39%, which is reasonable considering the reduction in dimensions. This reduction in parameters also leads to a decrease in the total training time. The table shows that the training time reduced from 464 s/E to 128 s/E, indicating a significant improvement in efficiency, with a reduction of up to 70% in the total time used. Regarding accuracy, the results demonstrate an increase of up to 78%. This increase is aligned with the theory that shrinking dimensions using the operator-N can enhance translation equivariance within each capsule. The performance improvement results for other datasets are presented in TABLE 2, showing an accuracy increase of around 1.81%. From these experiments, we can conclude that the optimal shrink size hyperparameter is $\mathtt{s} = 8$. It is observed that larger shrinkage sizes result in faster training times, but at the cost of slightly lower accuracy. Therefore, based on multiple experiments, we recommend using $\mathtt{s} = 8$ as the hyperparameter for achieving optimal performance in the modified CapsNet with the addition of the operator-N layer.

3.2.2. *Operator-N Selection.* We have determined that the optimal hyperparameter for dimensional shrinkage is $\mathtt{s} = 8$, using the Euclidean norm as the operator. Next, we conducted experiments on various operator-Ns, including $\mathtt{N}_1(\cdot)$ (12), $\mathtt{N}_2(\cdot)$ (13), and $\mathtt{N}_3(\cdot)$ (14). The comparison of test accuracy for these operators is presented in TABLE 3. Based on the obtained results, it can be concluded that the $\mathtt{N}_1$ operator generally exhibits higher test accuracy compared to the other operators, except for the Kuzushiji MNIST dataset. This could be attributed to the fact that the $\mathtt{N}_1$ operator normalizes the values using the Euclidean norm, resulting in a more consistent range of values within the interval $[0, 1)$. In terms of training time, all operators have similar times, ranging from 158-167 s/E, which represents an approximately 11% increase compared to the baseline operator, the Euclidean norm. Consequently, it can be concluded that the most optimal operator to be used as the activation function at the operator-N layer is $\mathtt{N}_1$.

TABLE 4. Final Comparison of results on MNIST Dataset. (M:millions, s/E:seconds/Epoch)

Model	**Params**	**Bencmarks**		
		Time (s/E)	Loss	Accuracy
Original-CapsNet	8.21M	469	0.02730	96.99%
Our-CapsNet	**6.92M**	**161**	**0.02243**	**98.66%**

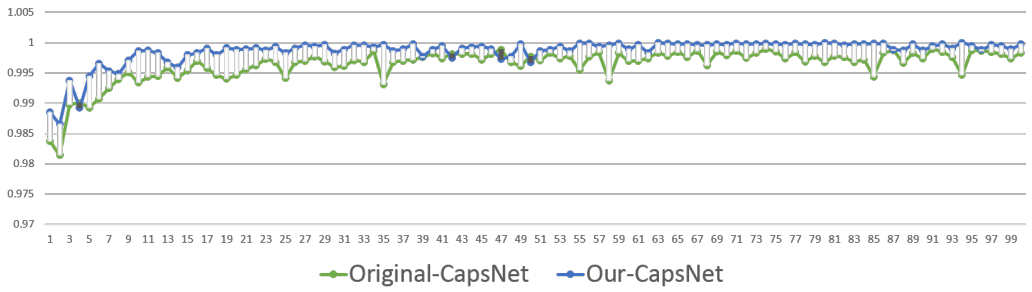

FIGURE 2. Curve of Training Accuracy on MNIST Dataset.

3.2.3. *Analysis of Performance.* We have determined that the optimal hyperparameter for the modified CapsNet method is dimension shrinkage with $s = 8$, using the $\mathtt{N}_1$ operator. We conducted additional training for 100 epochs to further analyze the performance while keeping other hyperparameters the same as before. FIGURE 2 illustrates the accuracy curves for both the Original-CapsNet and Our-CapsNet models during the training process. It is evident that Our-CapsNet exhibits a faster convergence rate and requires fewer epochs to reach convergence compared to Original-CapsNet. TABLE 4 illustrates the superior performance of Our-CapsNet compared to Original-CapsNet in all aspects. These findings align with the conclusions from the previous preliminary experiments, where Our-CapsNet achieved a reduction of 15.7% in parameters used, a decrease in the loss by 17.84%, and an increase in accuracy of 1.72%. One of the most notable improvements is observed in the total training time. Original-CapsNet required 469 s/E, equivalent to approximately 13.02 hours of training in real-time. On the other hand, Our-CapsNet only required 161 s/E, which is approximately 4.47 hours of training. This significant reduction in training time showcases the efficiency of Our-CapsNet. It achieves better accuracy and trains 2.91 times faster than Original-CapsNet, providing a substantial advantage. In conclusion, the experimental results depicted in FIGURE 3 demonstrate the superior performance of Our-CapsNet. It achieves a remarkable accuracy improvement of up to 2.15% while significantly reducing the computation time by approximately 2.88 times compared to the baseline method. This outcome highlights the effectiveness and efficiency of Our-CapsNet in enhancing accuracy and optimizing computational resources.

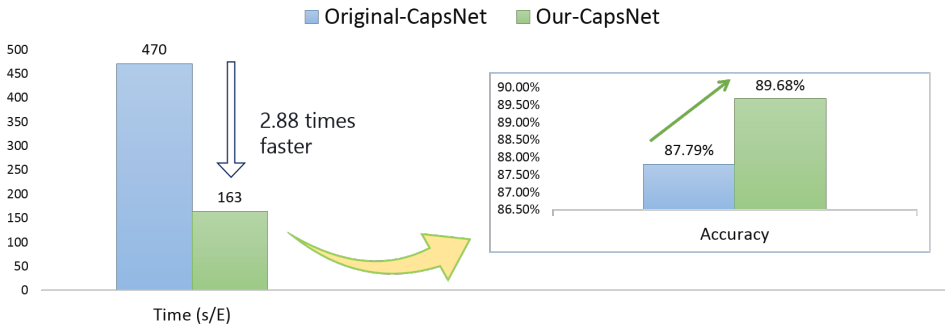

FIGURE 3. Increasing Accuracy with Computation Time on the Kuzushiji MNIST Dataset.

4. **Conclusion.** This paper presents a novel approach to enhance the Capsule Network method by incorporating an operator-N layer. The operator-N is an operator constructed using Euclidean norms, which functions to shrink dimensions and speed up the process of translation equivariance from a capsule. Through this modification, we observed significant improvements in various performance metrics. Specifically, the total parameters used were reduced by up to 15.7%, accompanied by a 17.84% decrease in loss and a 1.72% increase in accuracy. Furthermore, the training time was reduced by 2.91 times its original duration. The experimental results, conducted on the MNIST, Fashion MNIST, and Kuzushiji MNIST datasets, consistently demonstrated the effectiveness of integrating operator-N layer in enhancing the original Capsule Network's performance. In future work, we aim to extend the application of the operator-N layer to other deep learning methods to enhance their performance further.

Acknowledgment. This work was supported by Direktorat Riset, Teknologi, dan Pengabdian kepada Masyarakat, Kementerian Pendidikan, Kebudayaan, Riset dan Teknologi, in accordance Kontrak Induk Pelaksanaan Program Bantuan Operasional Perguruan Tinggi Negeri Penelitian Fundamental - Reguler Nomor Kontrak Induk: 112/E5/PG.02.00.PL/2023, dated June 19, 2023, Nomor Kontrak Peneliti: 1932/PKS/ITS/2023, dated June 20, 2023.

REFERENCES

[1] Y. Zhang and Z. Dong, Medical imaging and image processing, 2023.
[2] X. Li, J. Zhang, Y. Du, J. Zhu, Y. Fan, and X. Chen, A novel deep learning-based sentiment analysis method enhanced with emojis in microblog social networks, *Enterprise Information Systems*, vol. 17, no. 5, p. 2037160, 2023.
[3] S. Suparatpinyo and N. Soonthornphisaj, Smart voice recognition based on deep learning for depression diagnosis, *Artificial Life and Robotics*, pp. 1–11, 2023.
[4] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, Handwritten digit recognition with a back-propagation network, *Advances in neural information processing systems*, vol. 2, 1989.
[5] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*, 2014.
[6] 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 conference on computer vision and pattern recognition*, pp. 1–9, 2015.
[7] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[8] S. Sabour, N. Frosst, and G. E. Hinton, Dynamic routing between capsules, *Advances in neural information processing systems*, vol. 30, 2017.

[9] P. Peng, Z. He, L. Wang, and Y. Jiang, Microseismic records classification using capsule network with limited training samples in underground mining, *Scientific Reports*, vol. 10, no. 1, pp. 1–16, 2020.

[10] R. Mukhometzianov and J. Carrillo, Capsnet comparative performance evaluation for image classification, *arXiv preprint arXiv:1805.11195*, 2018.

[11] A. Mobiny and H. Van Nguyen, Fast capsnet for lung cancer screening, in *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II 11*, pp. 741–749, Springer, 2018.

[12] G. E. Hinton, S. Sabour, and N. Frosst, Matrix capsules with em routing, in *International conference on learning representations*, 2018.

[13] D. Wang and Q. Liu, An optimization view on dynamic routing between capsules, 2018.

[14] J. E. Lenssen, M. Fey, and P. Libuschewski, Group equivariant capsule networks, *Advances in neural information processing systems*, vol. 31, 2018.

[15] Y. Zhou, R. Ji, J. Su, X. Sun, and W. Chen, Dynamic capsule attention for visual question answering, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 9324–9331, 2019.

[16] K. N. R. Suyanto and S. Mandala, Deep learning modernisasi machine learning untuk big data, *Informatika*, 2019.

[17] K. Conrad, Equivalence of norms, *Expository Paper, University of Connecticut, Storrs, heruntergeladen von*, vol. 17, no. 2018, 2018.

[18] L. Deng, The mnist database of handwritten digit images for machine learning research, *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[19] H. Xiao, K. Rasul, and R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[20] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, Deep learning for classical japanese literature, 2018.