electronics

MDPI

Article

A Multi-Scale Convolutional Neural Network for Rotation-Invariant Recognition

Tzung-Pei Hong [1,2,*], Ming-Jhe Hu [3], Tang-Kai Yin [1] and Shyue-Liang Wang [4]

[1] Department of Computer Science and Information Engineering, National University of Kaohsiung, Kaohsiung 811726, Taiwan; tkyin@nuk.edu.tw
[2] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 804201, Taiwan
[3] Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701401, Taiwan; ryanhutech@gmail.com
[4] Department of Information Management National, University of Kaohsiung, Kaohsiung 811726, Taiwan; slwang@nuk.edu.tw
* Correspondence: tphong@nuk.edu.tw

Abstract: The Internet of things (IoT) enables mobile devices to connect and exchange information with others over the Internet with a lot of applications in consumer, commercial, and industrial products. With the rapid development of machine learning, IoT with image recognition capability is a new research area to assist mobile devices with processing image information. In this research, we propose the rotation-invariant multi-scale convolutional neural network (RIMS-CNN) to recognize rotated objects, which are commonly seen in real situations. Based on the dihedral group D4 transformations, the RIMS-CNN equips a CNN with multiple rotated tensors and its processing network. Furthermore, multi-scale features and shared weights are employed in the RIMS-CNN to increase performance. Compared with the data augmentation approach of using rotated images at random angles for training, our proposed method can learn inherent convolution kernels for rotational features. Experiments were conducted on the benchmark datasets: MNIST, FASHION-MNIST, CIFAR-10, and CIFAR-100. Significant improvements over the other models were achieved to show that rotational invariance could be learned.

Keywords: convolutional neural network; rotational invariance; multi-scale feature; dihedral group; weight sharing

Citation: Hong, T.-P.; Hu, M.-J.; Yin, T.-K.; Wang, S.-L. A Multi-Scale Convolutional Neural Network for Rotation-Invariant Recognition. *Electronics* **2022**, *11*, 661. https://doi.org/10.3390/electronics11040661

Academic Editor: Jun Dong Cho

Received: 29 December 2021
Accepted: 14 February 2022
Published: 21 February 2022

1. Introduction

The devices in Internet of Things (IoT) collect data from their surroundings using various sensors, cameras, and alarms. They can be installed to give round-the-clock security for smart homes, monitor livestock health for smart farming, and detect non-authorized access for industrial safety. Image data are pivotal in these applications, and pattern recognition from them is fundamental to the desired functions of IoT devices. The recent advances of deep learning for imaging processing and recognition have found many applications, including classification, detection, and segmentation of the image data from the devices.

Some important convolutional neural networks (CNNs) and extensions were provided for imaging recognition. In 1998, LeNet [1] was proposed with limited functions due to the computation ability of hardware at that time. These limits were gradually reduced along with the development of faster and more powerful GPUs. On the other hand, large labeled datasets such as ImageNet enabled CNNs to learn more complex and practical functions. Following LeNet, deeper and parallel convolution layers in CNNs were used in both AlexNet [2] and VGG [3]. These better structures improved recognition accuracy greatly. The number of network layers in CNNs was further increased in VGG16, in

which 13 convolutional layers and 3 fully connected layers were used to make a 16-layer CNN. The features were extracted mainly by three 3 × 3 convolution layers and one 2 × 2 max-pooling layer.

Deeper CNNs have greater computing power, but their training becomes difficult due to vanishing gradients and overfitting. To overcome this drawback, residual learning with shortcuts in a residual block was designed in ResNet-v1 [4]. Further modifications were then added in ResNet-v2 [5] to reshuffle the permutation of each layer in the residual block with better performance. In addition to deeper structures, network breadth was increased in the Google Inception series [6,7], which used multi-size convolution kernels within the same level of feature extraction. DenseNet [8] then increased the direct connections across layers to let each layer receive direct signal transmissions from all previous layers. This could partially solve the problem of vanishing gradients.

The CNN preserved the translation equivalence due to the convolution and pooling operations but did not keep the rotational invariance [9,10]. Hence, when a CNN is used to classify rotated images not reflected in the training data, the model accuracy considerably drops. For handling rotated images, many approaches were proposed to improve the rotation-invariant property of CNNs. In [11], only one-orientation training data were used to train CNNs. The N rotational versions of one original image were sequentially input into the well-trained model to obtain N rotational feature sets, which were then stacked on each other. The element-wise max-pooling operation was then performed on corresponding positions to form a responding set. In [12], Kang used rotation-augmented training data to train the CNN to achieve a higher wafer-map classification performance. In [13,14], instead of using data augmentation from rotational images for training, rotating features were learned from the rotated groups of images in the same training batch. In [15], rotation equivalence was achieved via four operations on the original feature maps: slice, roll, stack, and pool, while in [16–18], rotatable or rotation-invariant filters were employed. In [19], Kim et al. adopted cylindrical sliding windows in a convolutional layer to map the image into a polar coordinate system for achieving rotational invariance. Unlike these approaches, in this research, we seek to create a stable, rotation-invariant model that uses one-stage training and achieves high classification accuracy for general images.

2. Related Work

2.1. Rotation Invariance and the Dihedral Group D_4 (Dih_4)

When a vector is mapped to a rotational feature space and then input into a feature-extraction function, if the resulting features are the same as those obtained by directly applying the function on the original vector, the function is said to be rotation-invariant [20]. Figure 1 shows the concept. Here, we try to find a feature-extraction function in the CNN to satisfy the above constraint.

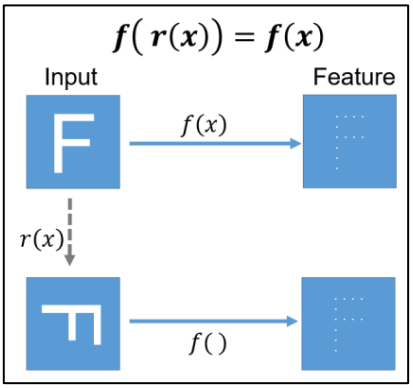

Figure 1. Rotational invariance.

A dihedral group D_n [21] is the symmetry group of a regular polygon with n sides and the rotation degree θ, equal to $360°/n$. When $n = 4$, the symmetry group (Dih_4) is

a square shape with all 90-degree rotations and flip operators. There are eight types of transformations, as shown in Figure 2.

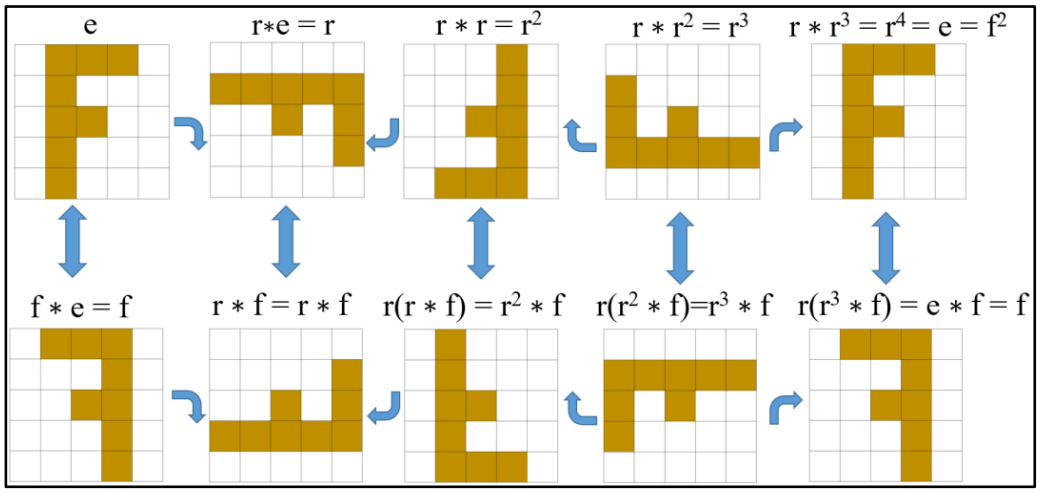

Figure 2. Dih_4 transformations.

In Figure 2, the identity element e in this group is the unchanged transformation, which does nothing with the regular polygon (the 5×5 square in Figure 2). A rotation transformation r rotates the regular polygon 90 degrees clockwise, and a flip transformation f flips the polygon about the vertical axis. Finally, a composition transformation rf first flips and then rotates a polygon. The binary operation (*) of a dihedral group is to composite these elements, which are e, r, f, and rf. Thus, the group Dih_4 must satisfy the following properties:

$$\begin{gathered} \forall r,\ f \in D,\ h = (r * f) \in D, \\ \forall r,\ f, h \in D,\ r(f * h) = (r * f)h \in D, \\ \exists e \in D,\ \forall h \in D,\ e * h = h, \\ \forall h \in D,\ \exists h^{-1} \in G,\ s.t.\ h * h^{-1} = h^{-1} * h = e. \end{gathered}$$

In this paper, we, thus, combine the Dih_4 symmetry transformation into the CNN to increase rotational invariance.

2.2. Multi-Scale Learning

The Single Shot MultiBox Detector (SSD) [22] is mainly used to detect target objects in images. It consists of a pretrained VGG16 feature extraction layer and a multi-scale CNN subnet. As shown in Figure 3, the convolutional layers at different depths are used to extract various scale features for bounding-box regression prediction, which is used in classification and object-location detection.

DenseNet [8] also presents the similar concept of passing all previous feature maps from each convolution block as inputs to the current convolution layer. This approach reuses all feature maps extensively to reduce the required weights, thus simplifying the model structure.

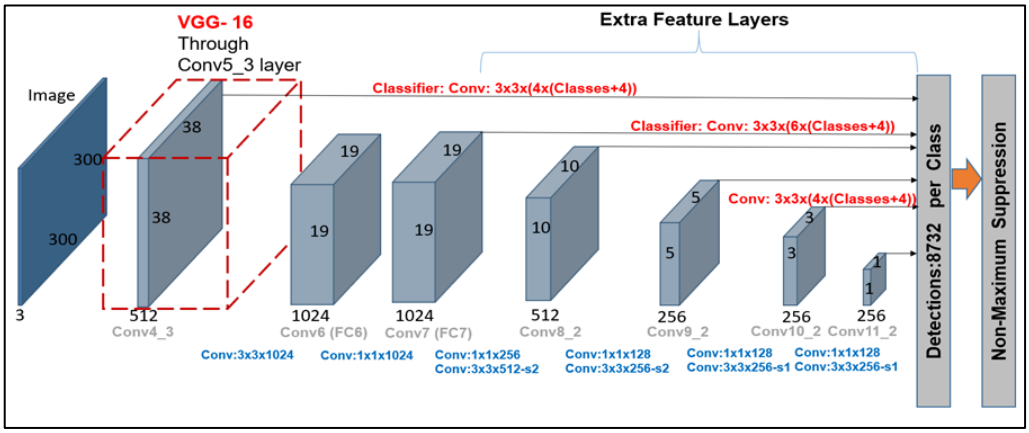

Figure 3. SSD architecture.

2.3. TI-Pooling CNN

The TI-pooling CNN [13], based on LeNet, gives an idea of augmenting a training set by rotating images. The TI-pooling CNN rotates an image with different angles in a training batch and shares the same set of weights in the same batch. As shown in Figure 4, after each rotated image is convoluted, the most significant features from the feature maps for each angle are found by the final max-pooling. The filters of this model can learn a regular pattern from different angles of the same image. It also decreases repeated patterns in the convolution filters generated by standard data augmentation and successfully enhances the rotational invariance of CNN recognition.

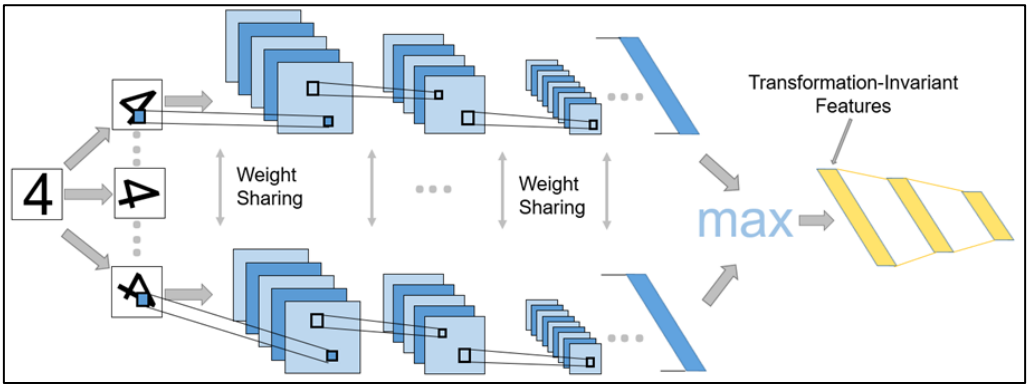

Figure 4. TI-POOLING CNN architecture.

3. Methods

For improving the classification accuracy of rotated objects, we propose the rotation-invariant multi-scale convolutional neural network (RIMS-CNN). The model is shown in Figure 5.

The model has three main parts: image transformation, feature extraction, and image classification. Image transformation is used to rotate an original image for generating images with different angles. Then, feature extraction is executed by four RI-Conv blocks, which are basic convolution units in our model, to capture different-scale image features. Finally, image classification combines the features in different directions and scales for recognizing objects. They are described below.

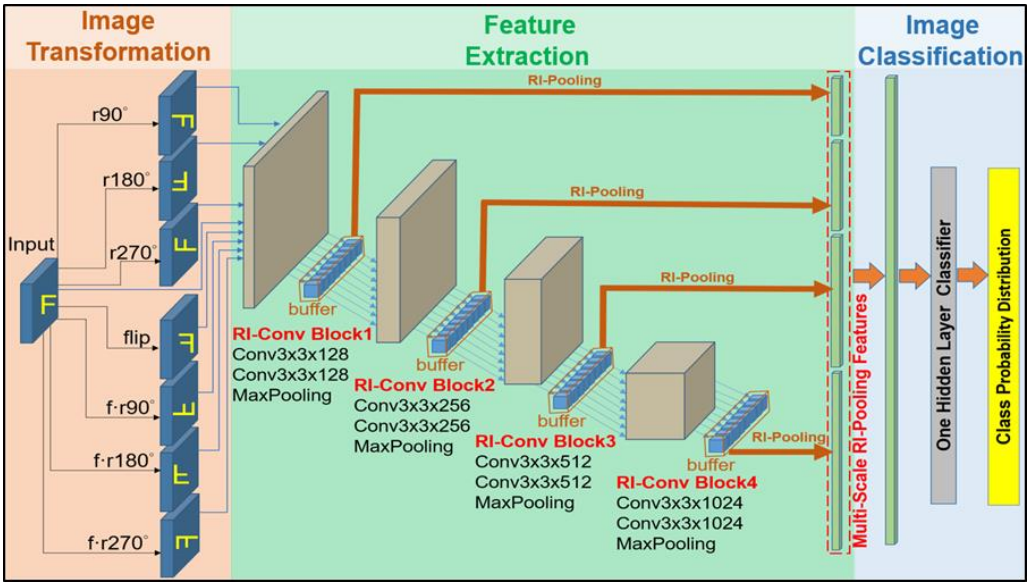

Figure 5. Proposed RIMS-CNN model.

3.1. Image Transformation

This module generates seven additional images from an original image by the Dih_4 operators as follows:

$$\begin{aligned} X^1 &= rotate90^\circ(X^0), \\ X^2 &= rotate180^\circ(X^0), \\ X^3 &= rotate270^\circ(X^0), \\ X^4 &= flip(X^0), \\ X^5 &= rotate90^\circ[flip(X^0)], \\ X^6 &= rotate180^\circ[flip(X^0)], \text{ and} \\ X^7 &= rotate270^\circ[flip(X^0)], \end{aligned}$$

where X^0 is an original input image, and X^1 to X^7 are the transformed ones from X^0. The transformation can increase the resistance to object rotation. We sequentially train the model on these transformed Dih_4 images and gather the features obtained from the RI-Conv blocks. The total loss of each training batch is, thus, highly related to the transformed Dih_4 images. Although traditional data augmentation also captures features from different transformations, the inputs are usually rotated randomly. The rotational inputs are not fed in one training batch, which causes the weights in the convolutional layers to change with the different feature spaces, thus increasing training time. In addition, the extracted rotational features can be redundant. Thus, this transformation strategy is more effective for learning rotational features than general data augmentation.

3.2. Feature Extraction

In this phase, we use several rotation-invariant convolution (RI-Conv) blocks to obtain implicitly important features. As shown in Figure 6, each RI-Conv block takes the output from the previous block as its input features, and then delivers its output to the next RI-Conv block and the rotation-invariant (RI) pooling module.

Since each transformed image is processed sequentially, a RI-Conv block collects eight feature groups and saves them in a buffer. After the eight sets of features are collected, they are sent from the buffer to the RI-pooling module as a subset of low-level features for final classification. Thus, the k-th RI-Conv block obtains the k-th-scale features. The feature maps in the buffer are also sent to the next RI-Conv block for further processing of higher-level features.

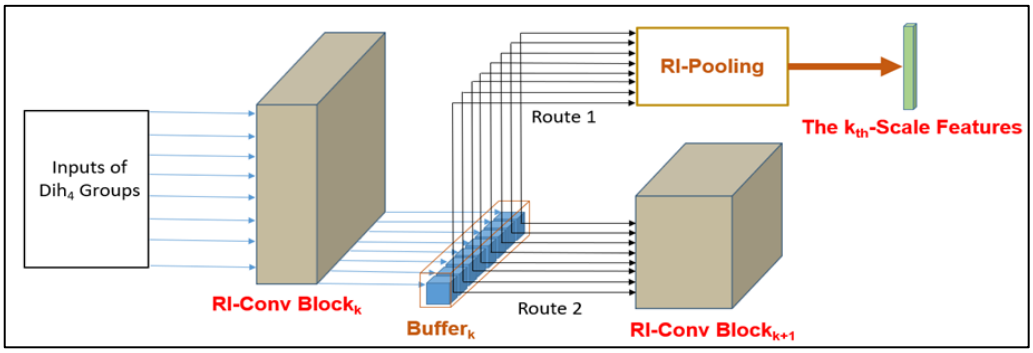

Figure 6. Proposed RIMS-CNN model.

3.2.1. RI-Conv Block Structure

RI-Conv blocks are designed for feature extraction. In our model, because the size of the input images is 28 × 28 or 32 × 32, we, thus, use four RI-Conv blocks, with the output size in the last RI-Conv block being 1 × 1 or 2 × 2. This can be easily extended for larger image sizes. Each RI-Conv block contains two convolutional layers and one max-pooling layer. Figure 7 shows the flowchart of an RI-Conv block.

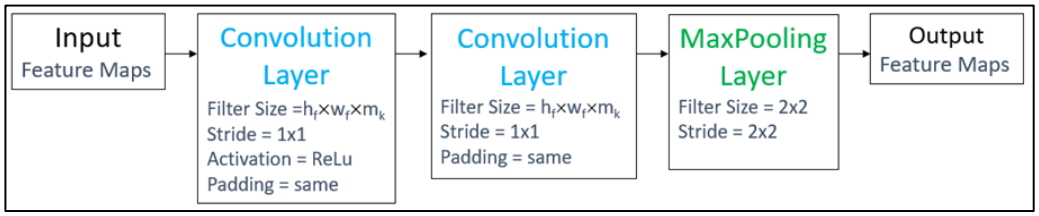

Figure 7. RI-Conv block flowchart.

3.2.2. Buffers in the RIMS-CNN Architecture

Buffers are used to store the output results of the RI-Conv blocks: each such block has a corresponding buffer, which includes eight components. Each component stores the set of feature maps from the respective Dih_4 transformed image. Figure 8 illustrates the buffer architecture, where F^{kg} denotes the group of feature maps generated from the g-th Dih_4 transformed image in the k-th RI-Conv block.

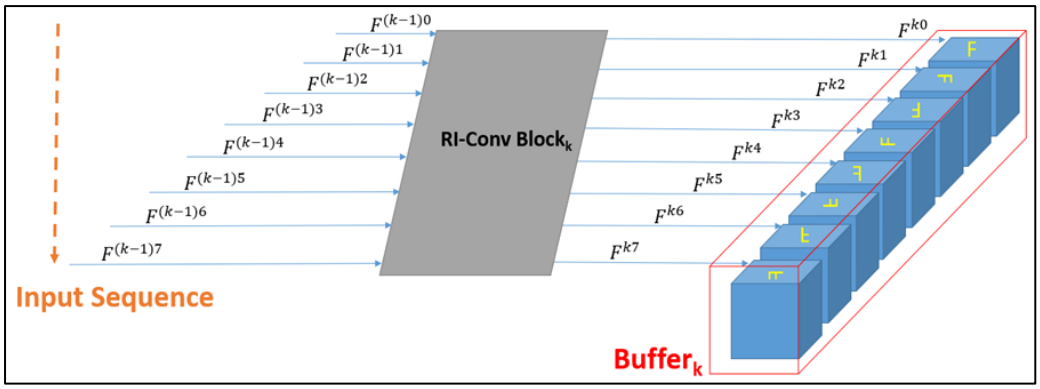

Figure 8. Buffer architecture following a RI-Conv block.

The buffer collects the eight feature groups—F^{k0} to F^{k7}—and sequentially sends them to the next RI-Conv block for further processing. At the same time, they are sent through another route (Route 1 in Figure 6) to the RI-pooling unit to be compressed as the set of features at the k-th scale. The features from all scales are finally used for image classification. Below, we explain how the RI-pooling unit works.

3.2.3. Rotation-Invariant Pooling (RI-Pooling) Unit

Figure 9 shows the operations of the RI-pooling unit. It has three procedures: stacking, reshaping, and average pooling. They are described below.

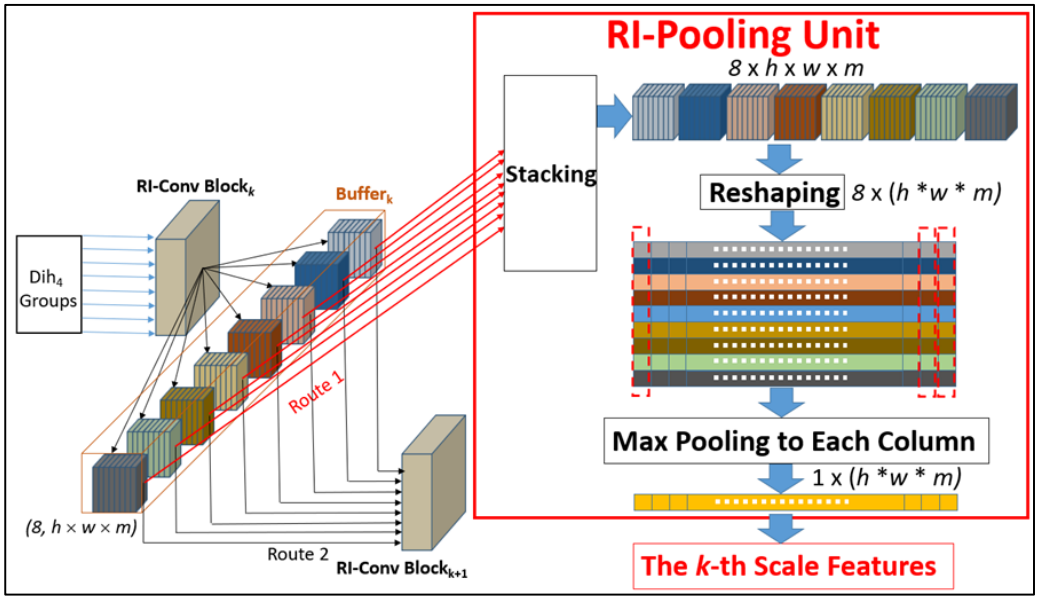

Figure 9. Three procedures of rotation-invariant pooling unit.

First, the stacking procedure stacks the outputs $\left\{F^{k0}, F^{k1}, \ldots, F^{k7}\right\}$ at the k-th RI-Conv block to form a 4-dimensional feature map, which is then reshaped to a 2-dimensional $8 \times (h \times w \times m)$ matrix. The i-th row, denoted R^{ki}, represents the group of the reshaped feature map generated from the i-th Dih_4 transformed image in the k-th RI-Conv block. The max-pooling procedure then processes each column to yield a feature value. Formally, for the j-th column, its max-pooling value (s_j^k) is calculated as:

$$s_j^k = maxpooling\left(r_j^{ki}\right),$$
$$0 \le i < 7,\ 0 \le j < h * w * m,$$

where r_j^{ki} is the j-th value of in R^{ki}. Thus, it generates a $1 \times (h \times w \times m)$ matrix to represent the k-th scalable feature set, denoted S^k. The pooling operation can significantly reduce the size. As the proposed architecture has a total of four RI-Conv blocks, four sets of rotation-invariant features—S^1 to S^4—are generated and then are concatenated.

3.3. Image Classification

Figure 10 shows the architecture of image classification. The resulting feature maps from the different scales are concatenated. Then, a fully connected neural network is used to classify images. The neural network consists of one dropout, one hidden layer, and one softmax layer.

Then, each P_q is compared to the true probability distributions and used to calculate the loss based on the categorical cross-entropy. After we ascertain the losses from a training batch, we use them to update the weights till the whole batches are trained. Our model uses the backpropagation algorithm to minimize the differences between the computed output and target value to update the model weights. Because the loss is obtained from different angles of an input image, the rotation-invariant property can be maintained.

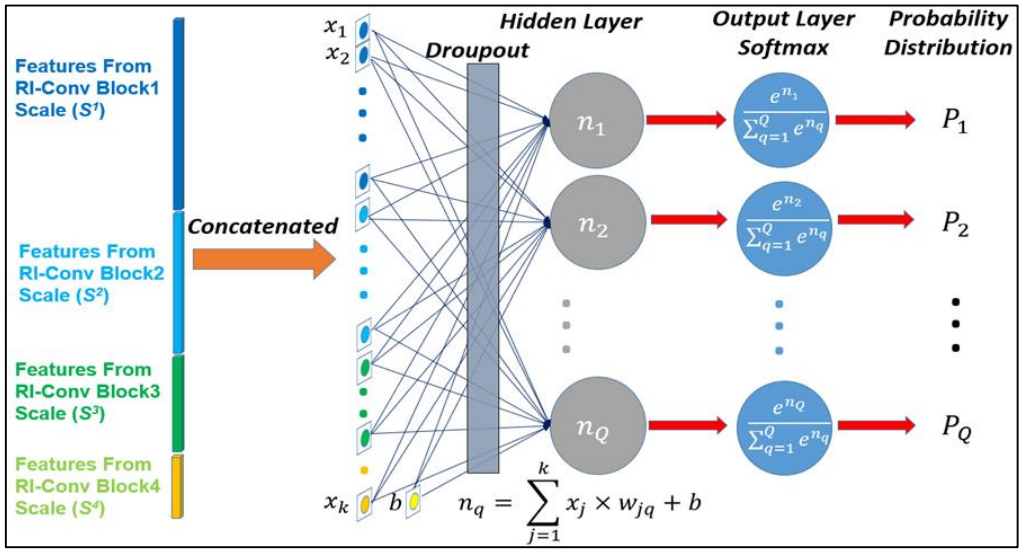

Figure 10. Image classification structure.

3.4. Time Complexity of RIMS-CNN

We first analyze the time complexity of processing one image. In the image transformation phase, assume d is the size of an image and c is the number of channels. Since the phase transforms the original image seven times, the time complexity is $O(7cd)$.

Then, we analyze the time complexity of the feature-extraction phase. An image in a particular direction goes through four RI-Conv Blocks, each of which comprises two convolution layers and a max-pooling layer. In the four blocks, let m be the maximum number of input (output) channels, j be the maximum size of the output feature map, and k be the maximum size of the filter. The time complexity is derived as $O(4(2 \times m \times j \times k \times m + j/4 \times m))$, where the first term is for the two convolutions and the second term is for the maximum pooling. Since the first term is larger than the second term, the time complexity can be simplified as $O(8 \times m \times j \times k \times m)$, which is $O(8m^2jk)$.

Because an image has eight transformed versions, the time complexity becomes $O(8(8m^2jk))$, which is $O(64m^2jk)$. Additionally, each RI-pooling unit includes stacking, reshaping, and column max-pooling functions. There are four RI-pooling units. Thus, the time complexity is $O(4(1 + 8 \times j \times m + j \times m))$, which is $O(36jm)$. The time complexity of the feature-extraction phase is, thus, $O(64m^2jk + 36jm)$, which is $O(64m^2jk)$.

The time complexity of the image classification phase is then analyzed. The time complexity of the fully connected layer is $O(4jmp)$, where p is the number of predicted classes and $4jm$ is the number of the multi-scale features.

The time complexities for processing an image can then be concluded as $O(7cd)$+ $O(64m^2jk)$+ $O(4jmp)$. Since the value of the second term dominates the other two terms, the time complexity can be simplified as $O(64m^2jk)$, which is $O(m^2jk)$ according to the big-O definition.

At last, assume the sample size is s and there are e epochs in the training process. The total time complexity is $O(esm^2jk)$. However, the time complexity can be reduced by the usage of GPUs since some steps can be executed in parallel.

4. Experimental Results and Discussion

We evaluated all the models on the MNIST [23], FASHION-MNIST [24], CIFAR-10 [25], and CIFAR-100 [25] datasets. Below we describe the datasets and the preprocessing.

4.1. Datasets and Preprocessing

Table 1 lists the metadata of the four datasets. MNIST contains single handwritten digits between 0 and 9, and FASHION-MNIST contains 10 fashion classes. Each image is grayscale, with the size of 28 × 28. In CIFAR-10, each image is colored, with the size of

32 × 32. There are 10 classes in the CIFAR-10 dataset. The last dataset, CIFAR-100, has 100 classes and has the same image numbers as the others. These four datasets are used to evaluate the performance of the proposed and the compared models. The preprocessing of the training and validating sets is described below.

Table 1. Four datasets details.

Dataset	Number of Training Set	Number of Testing Set	Image Size	Class Number
MNIST	60,000	10,000	28 × 28 × 1	10
FASHION-MNIST	60,000	10,000	28 × 28 × 1	10
CIFAR-10	60,000	10,000	32 × 32 × 3	10
CIFAR-100	60,000	10,000	32 × 32 × 3	100

In training-set preprocessing, since our proposed model is designed to handle rotational invariance, there is no need to augment the original training set with different angles. For other compared models, we preprocess the training sets in two ways: fixed-angle rotation and random-angle rotation. By fixed-angle rotation, each image is rotated with four angles: 0, 90, 180, and 270 degrees. Thus, the augmented training set is four times the size of the original. Figure 11 shows the concept of the fixed-angle rotation.

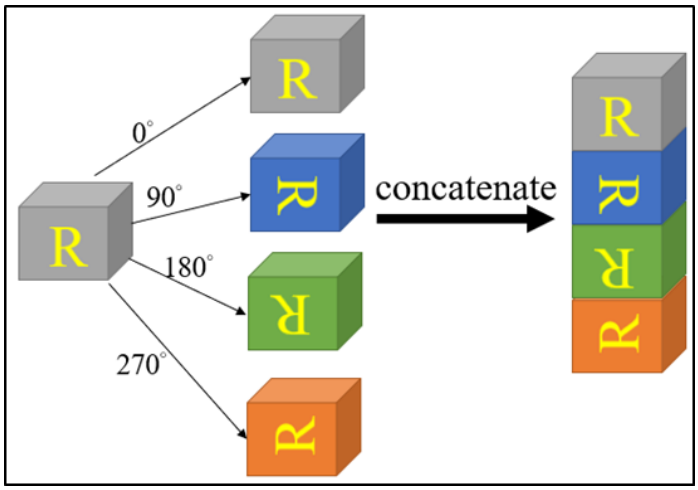

Figure 11. Fixed-angle rotation.

On the contrary, by random-angle rotation, each image is rotated with an arbitrary degree. The rotated training set is, thus, the same size as the original.

In training set preprocessing, we form three sets from the original testing set for evaluating the proposed and compared models. The first set is the original testing set with 10,000 images, the second set is the augmented testing set with the fixed-angle rotation, and the third set is the rotated testing set with the random-angle rotation. They are used to evaluate each model. For convenience, these sets will be called validation sets in the experiments.

4.2. Experimental Settings

In all the experiments, the models used only one fully connected layer with the output nodes to judge the classes of a dataset. The Adam [26] optimizer was adopted to learn the weights. The learning rate was set at 0.0001, the batch size was set at 64, and 1000 epochs were used in our experiments. Moreover, we used subsets of the datasets instead of all of the data to speed up weight convergence.

We used early stopping to improve accuracy and speed up model training [27]. If the accuracy of the testing dataset did not improve in the subsequent 30 epochs, we stopped the training process and saved the weights of the best training result.

4.3. Experimental Results

Table 2 shows the instance numbers of the classes in the four datasets. It could be observed that the classes in each testing set are balanced, so we use accuracy as our evaluation metric, which is defined as follows:

$$\text{Accuracy} = \frac{\text{number of correctly classified examples}}{\text{number of examples}}.$$

Table 2. The class number of the testing sets.

Dataset	Class	Number	Dataset	Class	Number	Dataset	Class	Number
MNIST	1	980	FASHION-MNIST and CIFAR-10	1	1000	CIFAR-100	1	100
	2	1135		2	1000		2	100
	3	1032		3	1000		3	100
	4	1010		4	1000			
	5	982		5	1000		⋯	⋯
	6	892		6	1000			
	7	958		7	1000		97	100
	8	1028		8	1000		98	100
	9	974		9	1000		99	100
	10	1009		10	1000		100	100

In our experiments, we used early stopping and recorded the best validation accuracy of each model as our evaluation value. Below are the experimental results for the different scenarios.

4.3.1. Different Design of the Feature Extraction Layers

We set the number of filters as 32, 64, 128, and 256 for the first to the fourth convolution blocks, respectively. Then, we used the original training sets and the fixed-angle rotating validating sets to evaluate the rotation-invariant improvements of applying our methods, which were Dih_4 transformation and multi-scale RI-pooling (MSRI) for the simple CNN. We also compared the accuracy of the 3×3 and 5×5 filters, which were used in the convolution layers. The results are shown in Figures 12–15, in which "single" denotes only one CNN layer used in a convolution block, and "average" and "max" denote the different integrated methods of the RI-pooling units.

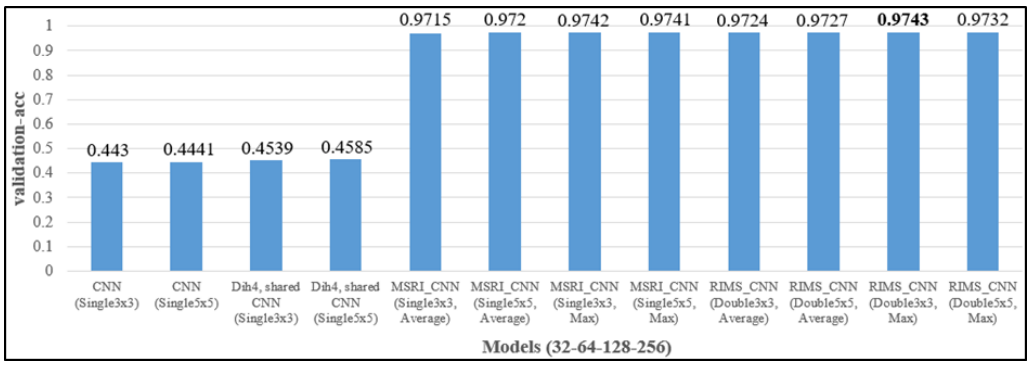

Figure 12. The results of different designs validated on the fixed-angle-rotated MNIST.

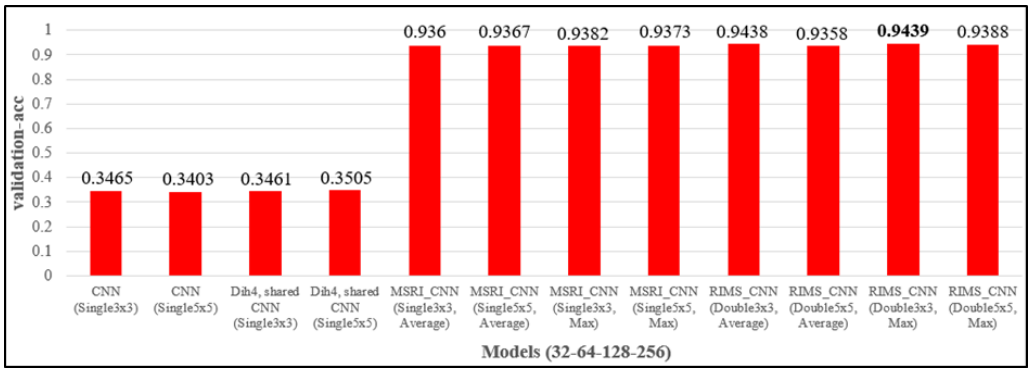

Figure 13. The results of different designs validated on the fixed-angle-rotated FASHION-MNIST.

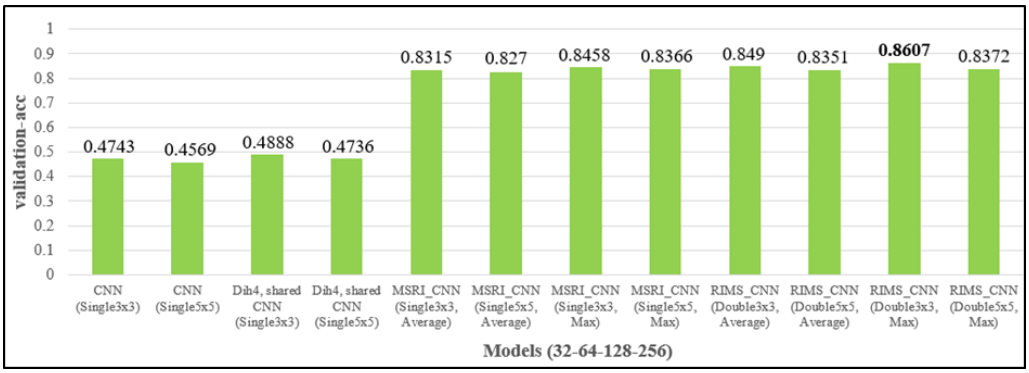

Figure 14. The results of different designs validated on the fixed-angle-rotated CIFAR-10.

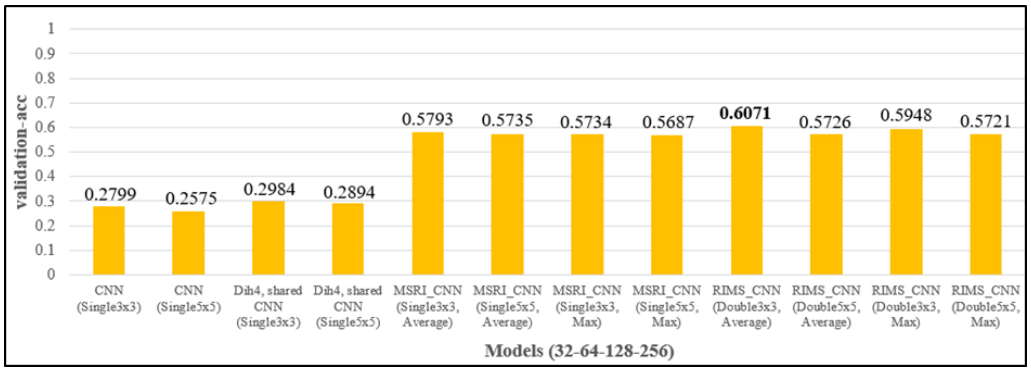

Figure 15. The results of different designs validated on the fixed-angle-rotated CIFAR-100.

The results from the first four models in the figures show when the Dih_4 transformation was applied on a shared CNN, the accuracy was only slightly improved. However, after we applied the multi-scale RI-pooling units, the accuracy was hugely increased. The results also show that our model using 3×3 filters in convolution layers mostly had better results than using 5×5 filters. According to the results, we used the RIMS-CNN with 3×3 filters as our primary model to compare the proposed model with the others in the following experiments.

4.3.2. Comparing RI-Max Pooling with RI-Average Pooling in Different Filters of RIMS-CNN

According to the previous subsection result, we selected the RIMS-CNN as our base model and compared the RI-max pooling with RI-average pooling in the RIMS-CNN, which was set with the different number of filters in RI-conv blocks. We tried to determine which pooling can capture better rotation-invariant features and find the best filter setting for the

RIMS-CNN. The training sets are original, and the validating sets are fixed-angle rotations. The results are shown in Figures 16–19.

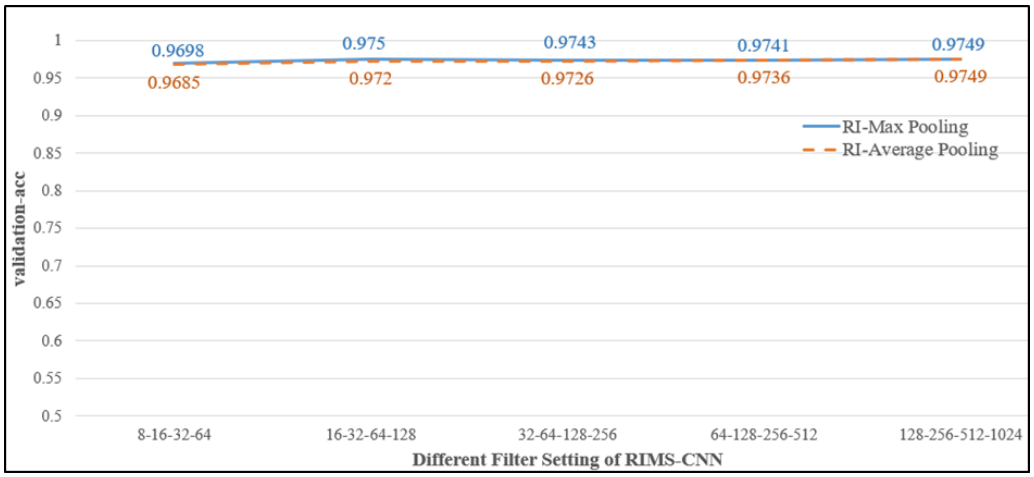

Figure 16. Comparing RI-max pooling with RI-average pooling on the fixed-angle rotated MNIST.

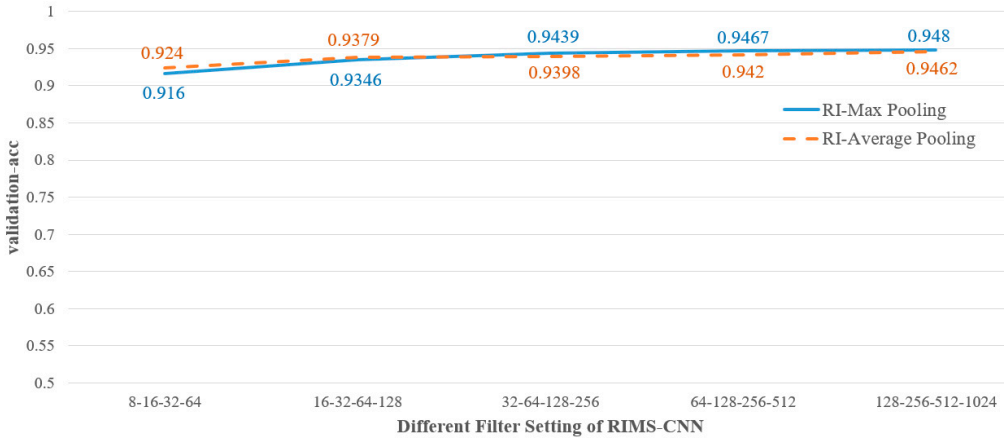

Figure 17. Comparing RI-max pooling with RI-average pooling on the fixed-angle rotated FASHION-MNIST.

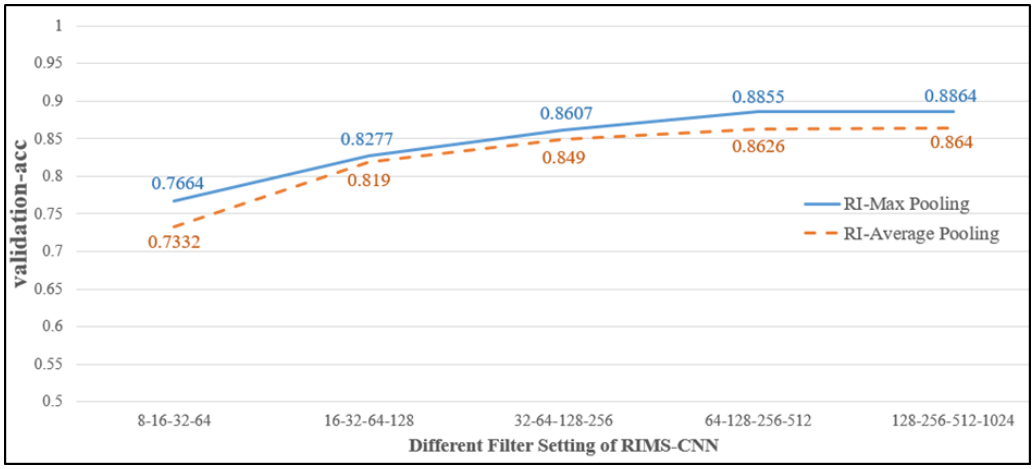

Figure 18. Comparing RI-max pooling with RI-average pooling on the fixed-angle rotated CIFAR-10.

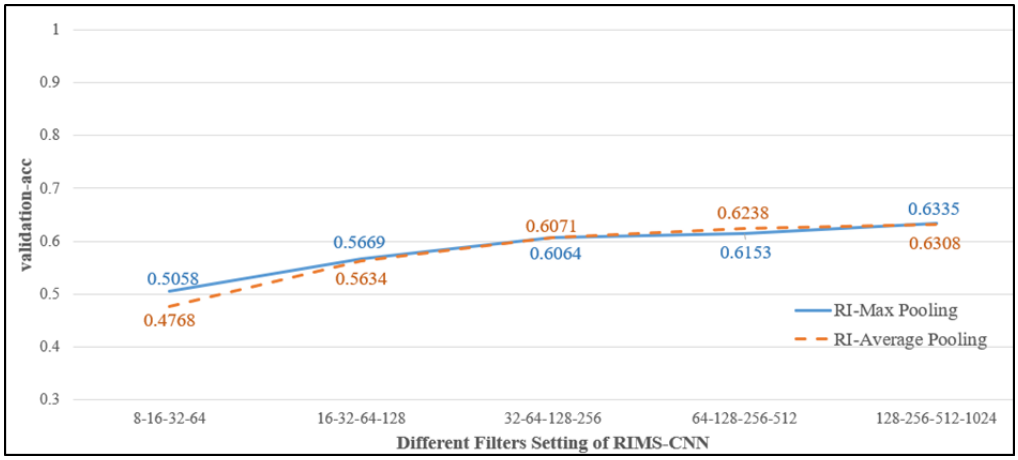

Figure 19. Comparing RI-max pooling with RI-average pooling on the fixed-angle rotated CIFAR-100.

In Figure 16, we can see that the accuracies of RI-max pooling are slightly better than RI-average pooling. Figure 17 shows that the RI-average pooling performs better in the 8-16-32-64 and 16-32-64-128 filter settings, but not by a big difference. In both pooling, the number setting of the 128-256-512-1024 filter has the best accuracy.

In Figure 18, the RI-max pooling has the better performance, but in Figure 19, the RI-max pooling does not win all filter settings. Over the four datasets, the RI-max pooling with the 128-256-512-1024 filter setting has the best accuracy. We decided to use the 128-256-512-1024 filter setting and further compared the RI-max pooling and RI-average pooling with different validating sets in the next experiment.

4.3.3. Comparing RI-Max Pooling with RI-Average Pooling in Different Datasets

In this experiment, we used either the RI-max pooling or RI-average pooling in the RIMS-CNN. We compared both pooling in the four datasets, which had original, fixed-angle rotation, and random-angle rotation. Figures 20–23 show the same results: using RI-max pooling is better than RI-average pooling in the RIMS-CNN. Therefore, we used the RI-max pooling and 128-256-512-1024 filter number as our standard-setting. We then validated the different structures with the same filter number in the following experiment.

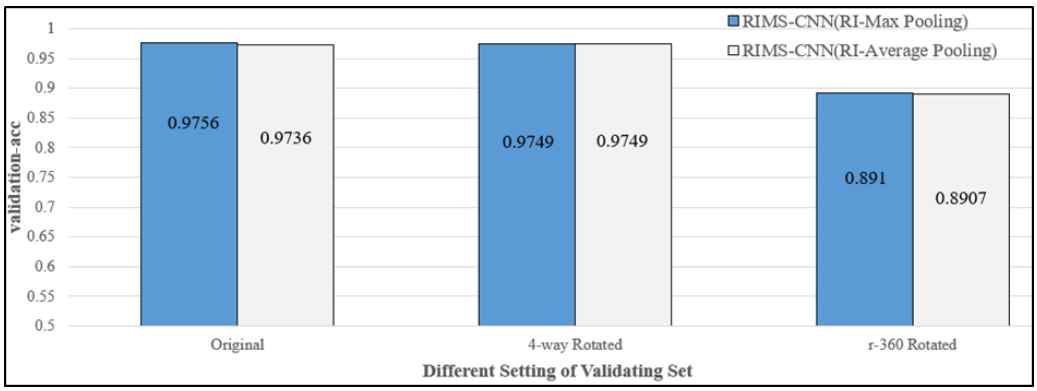

Figure 20. Comparing RI-max pooling with RI-average pooling in the different MNIST settings.

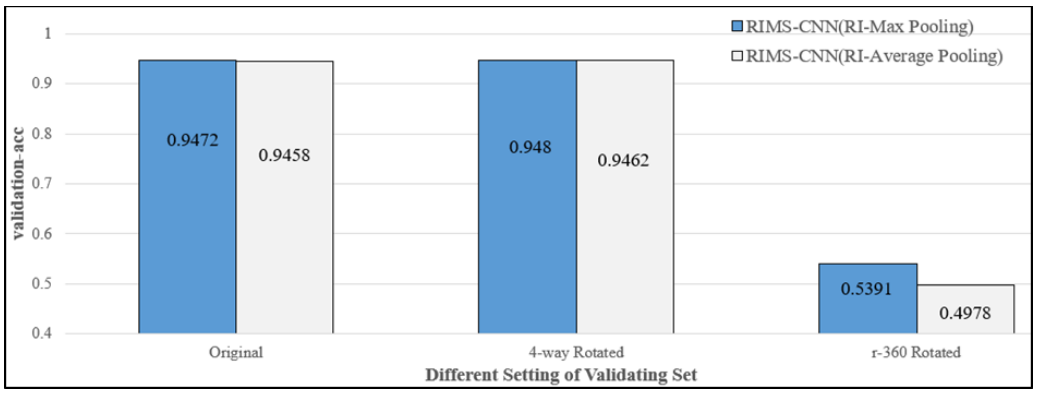

Figure 21. Comparing RI-max pooling with RI-average pooling in the different FASHION-MNIST settings.

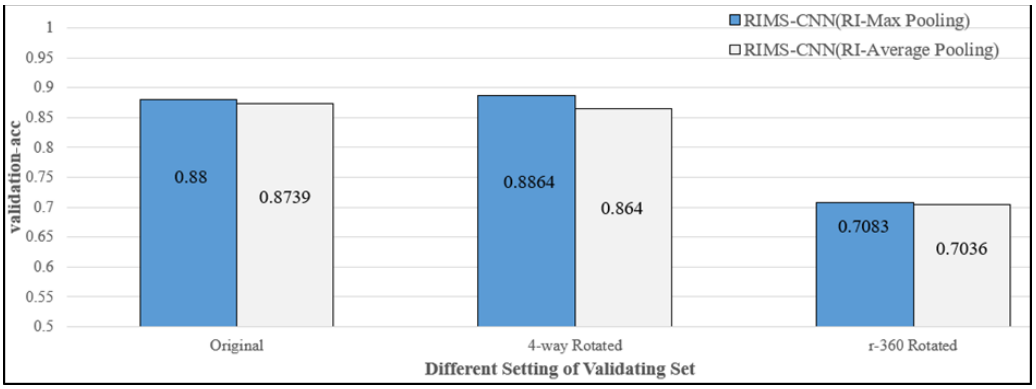

Figure 22. Comparing RI-max pooling with RI-average pooling in the different CIFAR-10 settings.

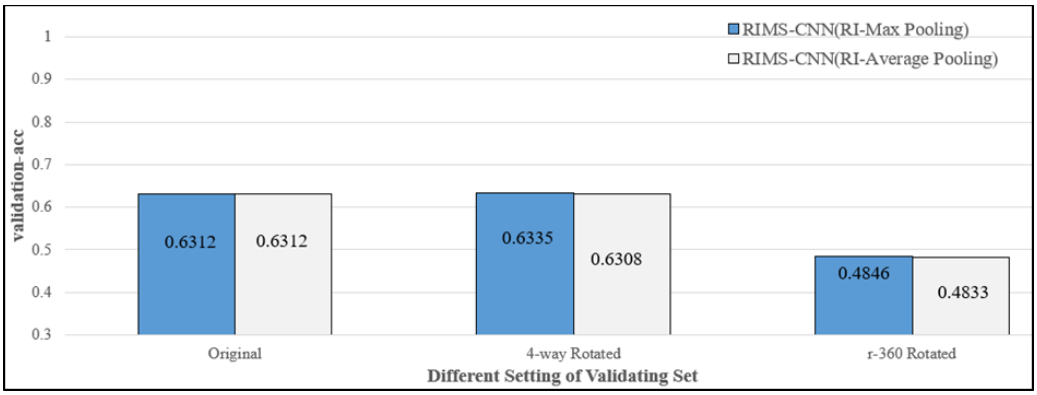

Figure 23. Comparing RI-max pooling with RI-average pooling in the different CIFAR-100 settings.

4.3.4. Evaluating Different Feature-Extracting Structures in RIMS-CNN

We used different structures but with the same number of filter settings (128-256-512-1024) to evaluate the fixed-angle rotated validating sets. Figures 24–27 show the RIMS-CNN had the best accuracy in FASHION-MNIST, CIFAR-10, and CIFAR-100, but not MNIST; however, this had only a 0.004 difference from the best value. We also found that the RIMS-CNN with multi-scale RI-stacking, which is used to stack the eight direction features without pooling, was worse than the RIMS-CNN with multi-scale RI-pooling. Therefore, we cannot use all multi-scale features directly as final classification features. Too many features from each scale significantly drop the classification accuracies.

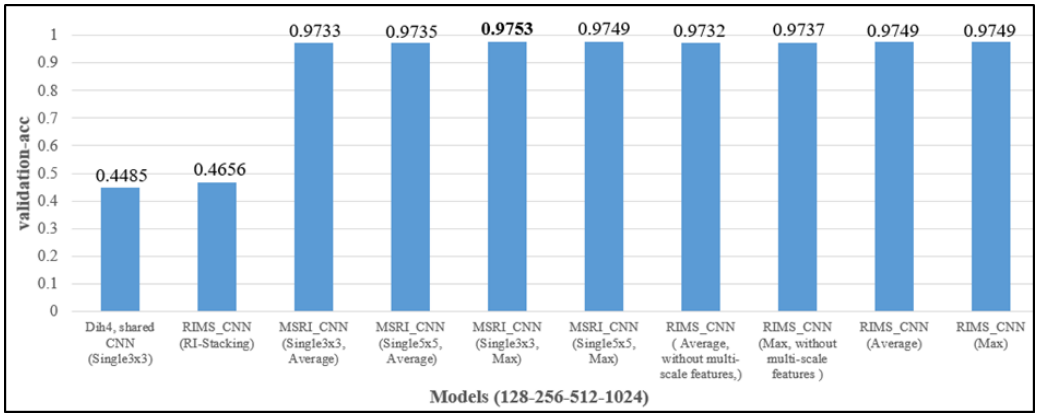

Figure 24. Evaluating different feature-extracting structures on the fixed-angle rotated MNIST.

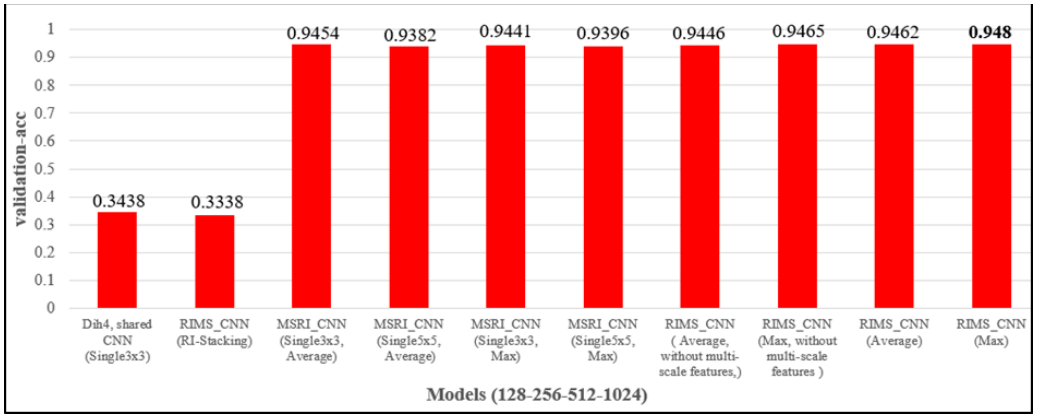

Figure 25. Evaluating different feature-extracting structures on the fixed-angle rotated FASHION-MNIST.

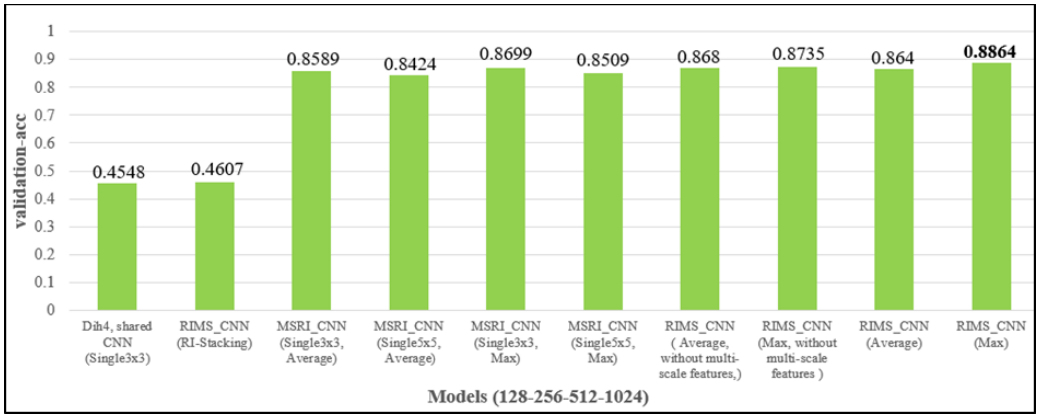

Figure 26. Evaluating different feature-extracting structures on the fixed-angle rotated CIFAR-10.

4.3.5. Evaluating Models on Original Training Sets and Validating Sets

In this experiment, we used the original training sets for model training and used the original validating set for evaluating the accuracy of each model. The evaluation results are shown in Table 3. The best results occurred in FASHION-MNIST, CIFAR-10, and CIFAR-100, but not MNIST. The results show that our model is effective.

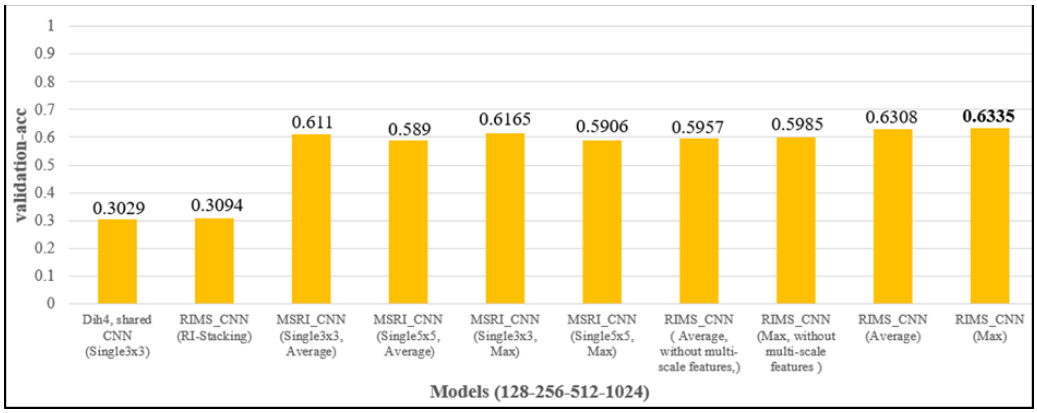

Figure 27. Evaluating different feature-extracting structures on the fixed-angle rotated CIFAR-100.

Table 3. Evaluating models on the original training and validating sets.

Models	Accuracy of the Original Validating Sets			
	MNIST	**FASHION-MNIST**	**CIFAR-10**	**CIFAR-100**
RIMS-CNN (Single FC Layer)	0.9756	**0.9472**	**0.88**	**0.6312**
DenseNet121 (Single FC Layer)	0.9962	0.9158	0.7635	0.4461
MobileNetV2 (Single FC Layer)	0.9952	0.9176	0.6887	0.3149
ResNet50V2 (Single FC Layer)	0.9962	0.9206	0.7467	0.4386
VGG-16 (Single FC Layer)	**0.9967**	0.938	0.8611	0.5267
InceptionV3 (Single FC Layer)	0.9962	0.9237	0.7742	0.405

4.3.6. Evaluating Models on Original Training Sets and Fixed-Angle Rotated Validating Sets

The original training sets were then input to train all the models. The fixed-angle rotated validating sets were used for evaluation. The results are shown in Table 4, in which the proposed model presented strong rotational invariance in the four datasets. The reason is that we embed the Dih_4 transformation and multi-scale RI features to learn rotational features in the proposed model. Other models did not effectively learn rotational features without rotating training data. Compared to the TI-pooling CNN designed for the transformation invariant problem, our model has better accuracy in FASHION-MNIST, CIFAR-10, and CIFAR-100.

4.3.7. Evaluating Models on Fixed-Angle Rotated Training Sets and Fixed-Angle Rotated Validating Sets

Here, we prepared the fixed-angle rotated training sets for each model to learn rotational features. The number of data was four times the size of the original sets. Table 5 shows that our model performed better than other models in FASHION-MNIST, CIFAR-10, and CIFAR-100. Compared to the results in Table 3, we know that if specific directional features were desired, the corresponding directional datasets could be generated to increase the accuracy.

Table 4. Evaluating models on the original training sets and the fixed-angle rotated validating sets.

Models	Accuracy of the Fixed-Angle Rotated Validating Sets			
	MNIST	**FASHION-MNIST**	**CIFAR-10**	**CIFAR-100**
RIMS-CNN (Single FC Layer)	0.9756	**0.9472**	**0.88**	**0.6312**
TI-Pooling (Single FC Layer)	**0.9799**	0.9229	0.7524	0.5042
DenseNet121 (Single FC Layer)	0.4475	0.3267	0.4373	0.2358
MobileNetV2 (Single FC Layer)	0.4329	0.3469	0.3945	0.1768
ResNet50V2 (Single FC Layer)	0.4166	0.3306	0.4187	0.2218
VGG-16 (Single FC Layer)	0.4438	0.3402	0.4645	0.2645
InceptionV3 (Single FC Layer)	0.3465	0.345	0.3057	0.1172

Table 5. Evaluating models on the fixed-angle rotated training sets and the fixed-angle rotated validating sets.

Models	Accuracy of the Fixed-Angle Rotated Training Sets			
	MNIST	**FASHION-MNIST**	**CIFAR-10**	**CIFAR-100**
RIMS-CNN (Single FC Layer)	0.9756	**0.9472**	**0.88**	**0.6312**
TI-Pooling (Single FC Layer)	0.9799	0.9229	0.7524	0.5042
DenseNet121 (Single FC Layer)	0.9954	0.9188	0.7692	0.4564
MobileNetV2 (Single FC Layer)	0.9936	0.9196	0.6903	0.3563
ResNet50V2 (Single FC Layer)	0.9953	0.9223	0.75	0.4128
VGG-16 (Single FC Layer)	**0.9956**	0.9398	0.8442	0.5375
InceptionV3 (Single FC Layer)	0.9951	0.9232	0.2046	0.0325

4.3.8. Evaluating Models on Fixed-Angle Rotated Training Sets and Random-Angle Rotated Validating Sets

We used the fixed-angle rotated training set to evaluate all the models on the random-angle rotated validating data. The results in Table 6 show that our method was suitable for the fixed-angle rotated validating sets and performed better than the other models in the random-angle rotated validating data. However, we can see that our model did not perform as well as the other models in the MINIST and FASHION-MNIST datasets. This shows that our model might increase misclassification probability in different classes with the same rotating features, such as the single-digit numbers 6 and 9.

4.3.9. Evaluating RIMS-CNN and VGG-16 on Fixed-Angle Rotated Testing Sets and Random-Angle Testing Sets

According to the experimental results in Sections 4.3.3 and 4.3.4, we can see that the validating accuracy of VGG-16 was close to RIMS-CNN. Therefore, we further split each training set into 75% training data and 25% validating data to train the RIMS-CNN and VGG-16 models. Then we evaluated the models on the four testing sets, which were fixed-

angle rotation and random-angle rotation. We can see that RIMS-CNN is not as effective as VGG-16 in Figure 28.

Table 6. Evaluating models on the random-angle rotated validating sets.

Models	Accuracy of the Random-Angle Rotated Validating Sets			
	MNIST	**FASHION-MNIST**	**CIFAR-10**	**CIFAR-100**
RIMS-CNN (Single FC Layer)	0.891	0.5391	**0.7083**	**0.4846**
TI-Pooling (Single FC Layer)	0.8869	0.5338	0.6343	0.3898
DenseNet121 (Single FC Layer)	**0.9499**	0.4913	0.6167	0.3626
MobileNetV2 (Single FC Layer)	0.9293	0.4727	0.5685	0.2766
ResNet50V2 (Single FC Layer)	0.9461	0.4831	0.5844	0.3314
VGG-16 (Single FC Layer)	0.9481	0.494	0.6723	0.4167
InceptionV3 (Single FC Layer)	0.9217	**0.5435**	0.5363	0.2168

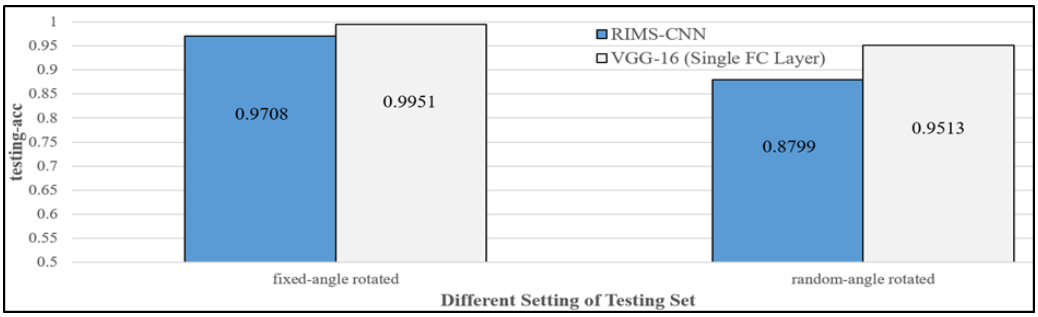

Figure 28. Evaluating RIMS-CNN and VGG-16 on the fixed-angle and random-angle rotated MNIST.

It still shows that our model may decrease accuracies on the image with highly symmetric rotation patterns, such as the single-digit numbers 6 and 9. However, Figures 29–31 show consistent results that RIMS-CNN is better than VGG-16.

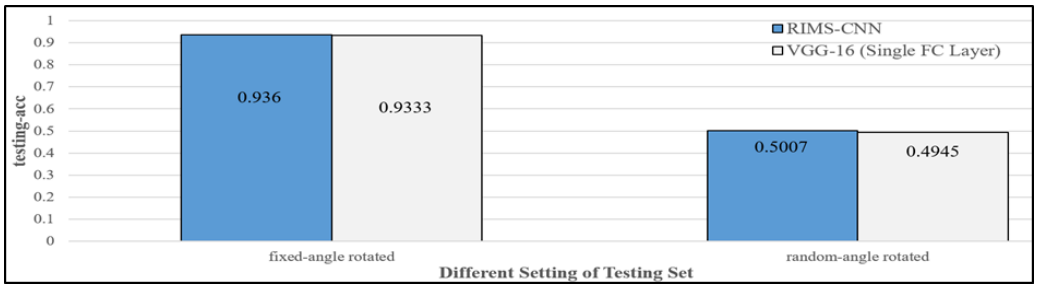

Figure 29. Evaluating RIMS-CNN and VGG-16 on the fixed-angle and random-angle rotated FASHION-MNIST.

4.3.10. Comparison of RIMS-CNN and Some Other Rotation-Invariant Models

Here we compare the results by RIMS-CNN and some other approaches which possess the property of rotation invariance. The results are shown in Table 7. These approaches are trained by the original training set and evaluated by the random-angle rotated validating data. We can see that all the accuracies from the approaches differed little for the MNIST

dataset, but that RIMS-CNN was the best. However, for CIFAR-10 and CIFAR-100, RIMS-CNN was significantly better than the other models. The results explain that our model has an excellent ability to learn the rotation invariance of complex images.

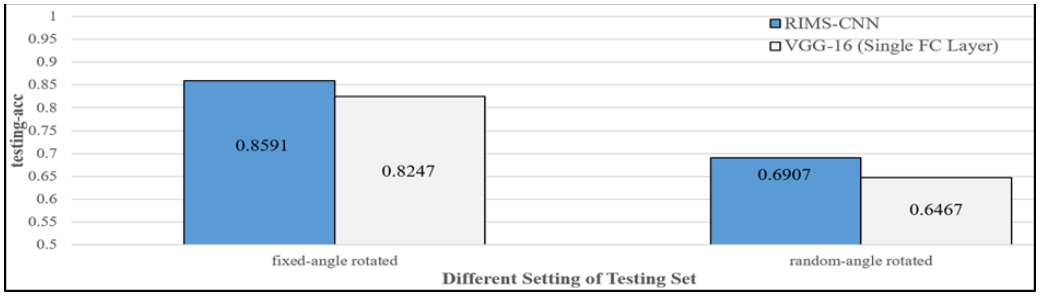

Figure 30. Evaluating RIMS-CNN and VGG-16 on the fixed-angle and random-angle rotated CIFAR-10.

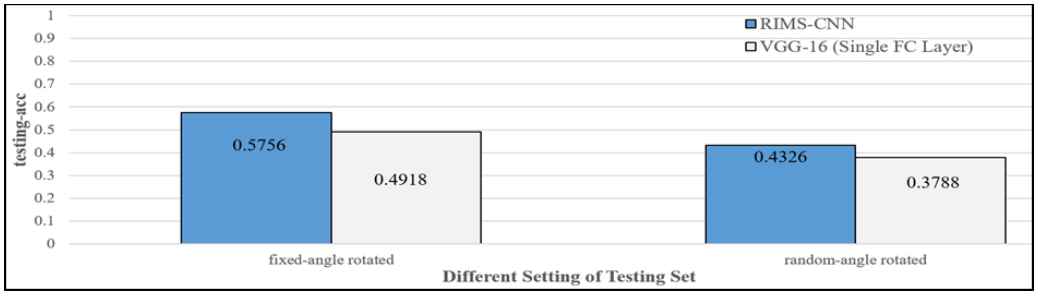

Figure 31. Evaluating RIMS-CNN and VGG-16 on the fixed-angle and random-angle rotated CIFAR-100.

Table 7. Comparison of RIMS-CNN and some other rotation-invariant models.

Model	Accuracy		
	MNIST	**CIFAR-10**	**CIFAR-100**
RIMS-CNN (Single FC Layer) (ours)	**0.891**	**0.7083**	**0.4846**
TI-Pooling (Single FC Layer)	0.8869	0.6343	0.3898
ORN-8 (ORPooling) [28]	0.8333	-	-
ORN-8 (ORAlign) [28]	0.8379	-	-
ORN [28]	-	0.4069	0.2164
RotInv Conv. (RP_RF_1) [29]	0.8015	-	-
RotInv Conv. (RP_RF_1_32) [29]	0.8780	-	-
RotInv Conv. (RP 1234) [29]	-	0.4412	0.2294
Covariant CNN [30]	0.8279	-	-

5. Conclusions and Future Work

In this paper, we attempt to increase the ability to identify rotated images in IoT applications. We have proposed three methods to achieve this purpose. First, the weights of all convolutional layers for a group of Dih_4 transformed images are shared to learn rotational features systematically. Second, RI-pooling is used to integrate eight related transformed groups of feature maps into one group. Third, multi-scale rotation-invariant (MSRI) features are used to increase the accuracy. The proposed model was compared with some other models on four datasets with three versions: the original set, fixed-angle rotation set, and random-angle rotation set. The experimental results show that the proposed model could outperform the others on accuracy when the training data was not augmented. Our proposed model still yielded superior performance, despite using data augmentation on FASHION-MNIST, CIFAR-10, and CIFAR-100 for the other models.

Object recognition with different rotation angles is usually requested for general applications such as traffic monitoring, self-driving image recognition, smart glasses, and

medical image recognition. The proposed method can effectively identify objects from different angles to fit the applications. It, thus, has the impact of adding the rotation invariance to the essential characteristics for actual application scenarios. Also, the Dih_4 transformation to an image needs to be done within the GPU memory. Although it can speed up the operation, the limitation is that it requires a larger amount of GPU memory.

In future work, we will focus on suitable extraction methods for symmetric images, such as those in MNIST. We will also study capturing features through the different spatial channels and design more complicated convolution blocks to extract higher-level features. In another aspect, we may try using a Generative Adversarial Network (GAN) to produce more diverse training data, not just limiting it to fixed or random angles. Finally, we would like to combine our proposed concepts with other state-of-the-art models and design other transforming strategies to embed in our model for improving the accuracy in rotation-invariant problems.

Author Contributions: Conceptualization, T.-P.H. and M.-J.H.; methodology, T.-P.H. and M.-J.H.; software, M.-J.H.; validation, T.-P.H., T.-K.Y. and S.-L.W.; formal analysis, T.-P.H. and S.-L.W.; investigation, M.-J.H. and T.-K.Y.; resources, M.-J.H.; data curation, M.-J.H.; writing—original draft preparation, M.-J.H. and T.-P.H.; writing—review and editing, T.-K.Y. and S.-L.W.; visualization, T.-K.Y.; supervision, T.-P.H.; project administration, T.-P.H. and S.-L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Science and Technology of the Republic of China under Grant MOST 109-2622-E-390-005 & MOST 110AO12B.

Data Availability Statement: MNIST: https://www.tensorflow.org/versions/r2.1/api_docs/python/tf/keras/datasets/mnist/load_data; FASHION-MNIST: https://www.tensorflow.org/versions/r2.1/api_docs/python/tf/keras/datasets/fashion_mnist/load_data; CIFAR-10: https://www.tensorflow.org/versions/r2.1/api_docs/python/tf/keras/datasets/cifar10/load_data; CIFAR-100: https://www.tensorflow.org/versions/r2.1/api_docs/python/tf/keras/datasets/cifar100/load_data.

Conflicts of Interest: This is a modified and expanded version of the paper "Extracting Multi-Scale Rotation-Invariant Features in Convolution Neural Networks", presented at the 2020 IEEE International Conference on Big Data. We also claim that this present study has not been published nor accepted for publication in other journals.

References

1. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
2. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the Neural Information Processing Systems (NIPS), Stateline, NV, USA, 3–8 December 2012; pp. 1097–1105.
3. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
4. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
5. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In Proceedings of the European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016; pp. 630–645.
6. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9.
7. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception architecture for computer vision. In Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
8. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
9. Gens, R.; Domingos, P.M. Deep symmetry networks. In Proceedings of the Neural Information Processing Systems (NIPS), Montréal, QC, Canada, 8–11 December 2014; pp. 2537–2545.
10. Lo, S.C.B.; Freedman, M.T.; Mun, S.K.; Gu, S. Transformationally identical and invariant convolutional neural networks through symmetric element operators. *arXiv* **2018**, arXiv:1806.03636.
11. Jain, A.; Sai Subrahmanyam, G.R.K.; Mishra, D. Stacked features based CNN for rotation invariant digit classification. In Proceedings of the Pattern Recognition and Machine Intelligence (PReMI), Kolkata, India, 5–8 December 2017; pp. 527–533.

12. Kang, S. Rotation-invariant wafer map pattern classification with convolutional neural networks. *IEEE Access* **2020**, *8*, 170650–170658. [CrossRef]
13. Laptev, D.; Savinov, N.; Buhmann, J.M.; Pollefeys, M. Ti-pooling: Transformation-invariant pooling for feature learning in convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 289–297.
14. Dieleman, S.; Willett, K.W.; Dambre, J. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Mon. Not. R. Astron. Soc.* **2015**, *450*, 1441–1459. [CrossRef]
15. Dieleman, S.; De Fauw, J.; Kavukcuoglu, K. Exploiting cyclic symmetry in convolutional neural networks. *arXiv* **2016**, arXiv:1602.02660.
16. Wu, F.; Hu, P.; Kong, D. Flip-rotate-pooling convolution and split dropout on convolution neural networks for image classification. *arXiv* **2015**, arXiv:1507.08754.
17. Marcos, D.; Volpi, M.; Tuia, D. Learning rotation invariant convolutional filters for texture classification. In Proceedings of the International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 4–8 December 2016; pp. 2012–2017.
18. Lo, S.B.; Freedman, M.T.; Mun, S.K.; Chan, H.-P. Geared rotationally identical and invariant convolutional neural network Systems. *arXiv* **2018**, arXiv:1808.01280.
19. Kim, J.; Jung, W.; Kim, H.; Lee, J. CyCNN: A rotation invariant CNN using polar mapping and cylindrical convolutional layers. *arXiv* **2020**, arXiv:2007.10588.
20. Woods, J.W. *Multidimensional Signal, Image, and Video Processing and Coding*, 2nd ed.; Academic Press: London, UK, 2011; pp. 28–30.
21. Judson, T.W. *Abstract Algebra: Theory and Applications*; Virginia Commonwealth University Mathematics: Richmond, VA, USA, 2009; pp. 81–83.
22. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; Berg, A.C. SSD: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016; pp. 21–37.
23. MNIST Handwritten Digit Database. Yann LeCun, Corinna Cortes and Chris Burges. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 17 December 2021).
24. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv* **2017**, arXiv:1708.07747.
25. CIFAR-10 and CIFAR-100 Datasets. Available online: https://www.cs.toronto.edu/~{}kriz/cifar.html (accessed on 17 December 2021).
26. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
27. Prechelt, L. Early stopping-but when? In *Neural Networks: Tricks of the Trade*, 2nd ed.; Müller, O., Ed.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 53–67.
28. Zhou, Y.; Ye, Q.; Qiu, Q.; Jiao, J. Oriented response networks. In Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 4961–4970.
29. Follmann, P.; Bottger, T. A rotationally-invariant convolution module by feature map back-rotation. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; pp. 784–792.
30. Salas, R.R.; Dokladalova, E.; Dokladal, P. Rotation invariant CNN using scattering transform for image classification. In Proceedings of the 2019 IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, 22–25 September 2019; pp. 654–658.