applied sciences

MDPI

Article

Accelerating Image Classification using Feature Map Similarity in Convolutional Neural Networks

Keunyoung Park and Doo-Hyun Kim *

Department of Software, Konkuk University, Seoul 05029, Korea; 981boxster@gmail.com
* Correspondence: doohyun@konkuk.ac.kr; Tel.: +82-2-2049-6044

Received: 23 November 2018; Accepted: 24 December 2018; Published: 29 December 2018

Abstract: Convolutional neural networks (CNNs) have greatly improved image classification performance. However, the extensive time required for classification owing to the large amount of computation involved, makes it unsuitable for application to low-performance devices. To speed up image classification, we propose a cached CNN, which can classify input images based on similarity with previously input images. Because the feature maps extracted from the CNN kernel represent the intensity of features, images with a similar intensity can be classified into the same class. In this study, we cache class labels and feature vectors extracted from feature maps for images classified by the CNN. Then, when a new image is input, its class label is output based on its similarity with the cached feature vectors. This process can be performed at each layer; hence, if the classification is successful, there is no need to perform the remaining convolution layer operations. This reduces the required classification time. We performed experiments to measure and evaluate the cache hit rate, precision, and classification time.

Keywords: image classification; convolutional neural network; feature map; cosine similarity

1. Introduction

Recently, artificial intelligence and machine learning research based on deep neural network technology has attracted significant attention. Owing to its high performance, it has been used in many fields such as image classification, detection, tracking, and natural language processing. In particular, convolutional neural networks (CNNs) have greatly enhanced the performance of image classification techniques. CNNs are composed of a very large number of layers. AlexNet [1], which was proposed in 2012, has five convolution layers; however, ResNet [2], with more than 100 convolution layers, was also developed recently.

As the number of layers increases, the amount of trained weights also increases. This requires a large amount of computation and thereby a significant computation time in the training and classification phase. The deployment of these networks in real-time applications is very difficult. To overcome these drawbacks, graphics processing units (GPUs) are used to speed up training and classification [3]; however, this requires a high-performance GPU or server. Therefore, it is difficult to apply CNNs to object classification that requires real-time property on a low-performance device with a low computing power. In other words, the acceleration of the classification speed is a key factor in practical applications and affects the resource utilization, power consumption, and latency [4].

We focus on reducing the execution time when classifying images using a CNN on low-performance devices. To speed up image classification using a CNN, Kang et al. [5] proposed a technique to make some computationally expensive layers work on a cloud server. This method can reduce the time required for classification; however, it has the disadvantage of always needing to be connected to the cloud server. Teerapittayanon et al. [6] proposed BranchyNet. BranchyNet

adds a branch at some point of the layer and ends the classification at this point, thus reducing the time required for classification. However, to use this method, new weights must be trained for new branches. Although sparse networks with network pruning and sparsity regularization have been studied [7–10], the weights must also be newly trained. That is, studies that modify the network structure cannot use a pretrained model, as the weight of each layer must be retrained.

To avoid retraining the pretrained CNN model, we propose a structure and method for caching feature maps that originate at each convolution layer. This structure is proposed because the devices used in the actual field receive only limited input. For example, an image classifier that can be mounted on an autonomous car receives only limited classes of input such as pedestrians, cars, bicycles, and traffic signs. Since the previously input image is likely to be input again, there is no need to perform all convolution layer operations every time. Therefore, images that are similar to previously entered images can be classified into the same image class.

With this structure, the class label can be obtained from the cache in the convolution layer operation of the input image; thereby reducing the classification time because the remaining convolution layers need not be operated. However, as there are so many convolution layers in a CNN, the image classification time can be reduced significantly by applying the cache structure mainly in the initial convolution layers.

Similar to our study, there are studies on CNNs that include caches. DeepCache [11] is used for mobile vision. If a block of a video frame matches a block of a previous frame, instead of recalculating the result, DeepCache gets the result of a similar block in the cache. However, we studied the classification of still images and found the similarities between input images and those that were classified in the same class, not the previous or next frame. In addition, after applying this cache to the pretrained CNN, the images may not be classified by the cache. In this case, the images can be classified through the CNN without using the cache; hence, a well-trained CNN can be used as is. To achieve this, we extract feature vectors from the feature maps generated in the convolution layers. In our proposed structure, the cache is activated after images classified by the CNN are written to the cache. However, there is no need to modify the CNN's structure or retrain the weights.

Feature maps are response images that are convolved by a trained kernel or filter and represent features extracted from the image. The CNN's kernel is similar to the Gabor filter [12]. In image classification, studies using intensity from various directions and frequencies extracted from the Gabor filter were used [13–15]. Likewise, the features extracted by the CNN kernel represent the intensity. For example, two completely different cat images can be classified as cats because the intensities of their eyes, nose, ears, and tail are similar.

Owing to the similarity of the CNN kernel and Gabor filters, many studies have been conducted that combined them. Sarwar et al. [16] used the Gabor filter in different layers of a CNN to reduce the computational complexity of the training phase. The Gabor filter was also used as a preprocessing method for weight training [17–19]. Luan et al. [12] replaced the Gabor filter with a convolution kernel in a CNN to perform the convolution and train the weights. In general, a CNN can be combined with a Gabor filter to achieve good performance.

In this study, we do not use the Gabor filter directly because we extract feature vectors from the pretrained CNN's feature map. However, given that the CNN kernel is generated by training, it does not extract certain features of the input image. This makes it difficult to describe or vectorize the feature maps. However, like the Gabor filter, which shows the intensity of the extracted features, we cache the input image using the intensity in the CNN's feature map.

Our main contribution is to locate the cache in the convolution layer of a CNN to make sure it is operable in order to improve the speed of image classification in low-performance devices. In particular, well-trained CNN models can be used without retraining. This structure is not a complete substitute for a CNN's image classification capabilities.

Therefore, this auxiliary structure assists the pretrained CNN model. When images corresponding to the same class are continuously input, the classes to which these images are classified are memorized and used when the next image is input.

2. Cached CNN Structure

In this section, we propose a cached CNN structure. Because a CNN consists of a large number of convolution layers, a large amount of time is required to classify input images and learn weights. However, assuming that the CNNs used in the actual field receive similar images consecutively, the input image can be classified by comparison with the previously input image, thus reducing classification time. Each convolution layer in a CNN generates feature maps from the input image, which represents the intensity of each feature. Input images with similar intensities can be classified in the same class. Therefore, we classify the input image by comparing its feature maps with the feature maps of the previously input images. To achieve this, feature vectors are extracted from feature maps generated from each convolution layer. Figure 1 shows this structure, which can be used to classify input images using a pretrained CNN model. It caches the feature maps of the images classified by the CNN, and classifies the newly input images by comparing them with the cached feature maps. This can be done at the convolution layer stage. If the classification is completed based on the similarity with the cached feature maps, the remaining CNN operations need not be performed. The steps of caching feature maps and image classification using them are as follows:

1. The input image is classified into its class by the pretrained CNN.
2. Feature vectors are extracted from the feature maps generated at each convolution layer, and they are written into the cache with a class label.
3. A new image is input for classification, and the CNN operation is begun. After the convolution layer operation is completed, a feature vector is extracted from the feature maps and compared to the feature vectors of the cached feature maps.
4. The class label of the most similar feature vector is output, and the remaining convolution layer operation is terminated.

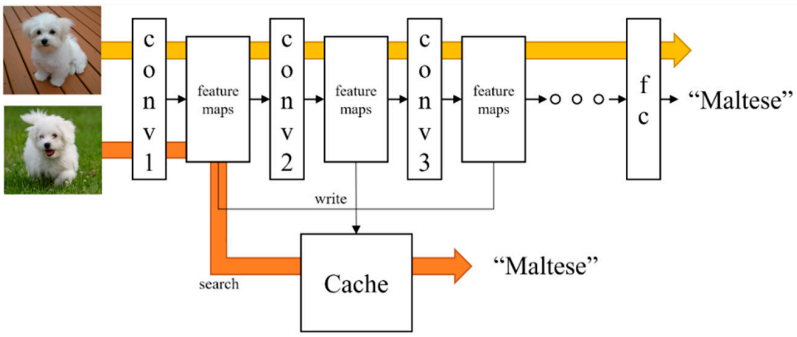


Figure 1. Proposed cached CNN structure. The yellow line indicates the flow in which the input image is classified by the CNN. The red line indicates the flow in which input image is classified by the cache.

3. Feature Map Caching Method

In this section, we present a method to extract a feature vector from feature maps and compare the query image with the previously input image by measuring their similarity. This method consists of two steps: (1) Feature vector extraction and (2) similarity measurement.

3.1. Feature Vector Extraction

Feature maps are a set of features obtained by convolving a trained kernel on each image. In addition, images that are classified into the same class have similar features. This is because the

intensity of each feature extracted from the images is similar. To generate a feature vector, this intensity can be expressed as energy. Therefore, for a feature map $F_i(x, y)$ of size $P \times Q$, we define the energy E as follows:

$$E(i) = \sum_x \sum_y |F_i(x, y)| \tag{1}$$

where i is the index of the feature map of each convolution layer.

From this, each feature map $F_i(x, y)$ is described by a mean μ_i, which represents the intensity of the individual pixels, and a standard deviation σ_i, which indicates the extent to which each pixel differs from adjacent pixels.

$$\mu_i = \frac{E(i)}{PQ} \tag{2}$$

$$\sigma_i = \sqrt{\frac{\sum_x \sum_y (|F_i(x, y)| - \mu_i)^2}{PQ}} \tag{3}$$

A feature vector V using the mean and standard deviation of feature maps for each convolution layer is generated as follows:

$$V = (\mu_0,\ \sigma_0, \mu_1, \sigma_1, \mu_2, \sigma_2, \ldots, \mu_{i-1}, \sigma_{i-1}) \tag{4}$$

The feature vector extracted above applies to both the step of writing the image classified from the CNN to the cache and of classifying the image using the cache. However, when writing classified images to a cache, it is undesirable to write feature vectors for all input images because the proposed method is used as a cache. Therefore, to have only one feature vector in one class for each convolution layer, images classified into the same class should have only one feature vector per convolution layer. That is, a representative feature vector corresponding to each class must be generated. Therefore, each time a new feature vector V_{new} is cached in the same class, the mean feature vector V_{cached} is updated as shown in Equation (5):

$$V_{cached} = \frac{(V_{cached} \times n) + V_{new}}{n + 1} \tag{5}$$

where n denotes the number of feature vectors accumulated for one class.

For this calculation, the cache also records the number of cumulative feature vectors for each class. Figure 2 shows this feature vector extraction process.

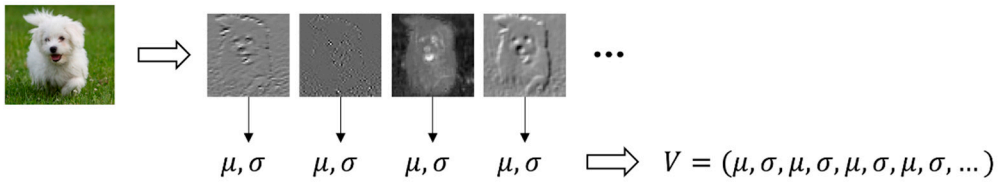


Figure 2. Feature vector extraction process. Mean and standard deviation of each feature map are calculated and listed to generate a feature vector V.

3.2. Similarity Measurement

The similarity of the input feature vector V_{query} (U) with the cached feature vector V_{cached} (T) can be defined as a cosine similarity. We use the following cosine distance to express similarity as distance:

$$D(U,\ T) = 1 - \frac{\sum_i (U_i \times T_i)}{\sqrt{\sum_i (U_i)^2} \times \sqrt{\sum_i (T_i)^2}} \tag{6}$$

This is a real value between 0 and 1. A distance value of zero means the two feature vectors are equal. From this, the input feature vector can be classified into the corresponding class of the cached feature vector with the highest similarity. However, in this case, the input feature vectors are classified as the most similar feature vectors among the cached feature vectors, which can lead to incorrect answers. Hence, if all input feature vectors are classified using only the cache, then all results will be a *Cache Hit*, which contains true and false positives. If a false positive is obtained, it is better to classify it using the CNN without the cache. Thus, even though the most similar cached feature vector is obtained for the input feature vector, this value should be set to a *Cache Miss* using the threshold value, if this result is a false positive.

Therefore, the threshold is defined as the cosine similarity between V_{new} (U) and V_{cached} (T) when V_{cached} is extracted.

This threshold is also updated every time the cache is accumulated. Therefore, the distance threshold D_{cached} for *Cache Hit* is defined as follows:

$$D_{cached} = \frac{(D_{cached} \times n) + D_{new}}{n+1} \tag{7}$$

where

$$D_{new} = 1 - \frac{\sum_i (U_i \times T_i)}{\sqrt{\sum_i (U_i)^2} \times \sqrt{\sum_i (T_i)^2}} \tag{8}$$

However, if the cumulative count of V_{cached} is one, the distance cannot be obtained. Therefore, to specify a threshold, at least two images in one class must be cached. Each time the cumulative count increases, a mean of the cached distance is refreshed. In addition, the cache of each convolution layer contains class, cumulative count ($\geq$2), mean distance, and mean feature vector.

Finally, when classifying the input image using the cache, the matching function f for determining *Cache Hit* from the input feature vector V_{query} and cached feature vector V_{cached} based on distance threshold ε is defined as:

$$f_\varepsilon(V_{query}, V_{cached}) \begin{cases} Hit & if \ \min\{D(V_{query}, V_{cached})\} \leq \varepsilon \\ Miss & otherwise \end{cases} \tag{9}$$

4. Experimental Results and Analysis

In this section, we measure and evaluate the performance of the proposed cached CNN structure. This experiment consists of three steps. First, we ensure that feature vectors extracted from feature maps in the same class have similarities. Second, we measure the hit rate of the cached CNN. Third, we measure the classification time of the cached CNN and compare it to the case where it is not applied. We also analyze the correlation between the hit rate and classification time. The program that operates all of the experiments uses Caffe [20].

4.1. Feature Vector Similarity

To verify that feature vectors of the same class have a similarity, we chose 10 classes from the Caltech256 [21] dataset and selected 100 images per class. The 100 images for each class were only images that were classified under the same class by AlexNet. The selected classes are {"gorilla," "backpack," "binoculars," "warplane," "joystick," "flower pot," "mountain bike," "toaster," "photocopier," and "daisy"}. Figure 3 shows an example of images for the 10 selected classes. Some of these images include those that were classified as the same class, although they were not, such as in "joystick" and "flower pot."

Figure 3. Examples of Caltech256 dataset images classified under the same class by AlexNet.

We then extracted feature vectors at each convolution layer of AlexNet and then measured the distance of each feature vector. The distance measurement involves two distances: (1) Distance between images of the same class. (2) Distance between images of different classes.

Distances between images of the same class were obtained by dividing the 100 images of each class into two groups of 50 each, and subsequently measuring the distance between the images of one group to the other. A total of $50 \times 50 = 2500$ distances were measured. Distances between images of different classes were determined by selecting 50 images for each class and measuring a total of $50 \times 50 = 2500$ distances. Figure 4 shows the results of this experiment. The horizontal axis of the graph represents the five convolution layers of AlexNet. The vertical axis represents a distance with a real value between 0 and 1. A red dashed-dotted line represents the distance between images of the same class.

In most cases, the distance between the images of the same class is the smallest. This is because the images classified into the same class have similar features generated by the convolution layer, and the intensity of the features is similar. In addition, it shows that our feature vector extraction method provides a discrimination that can distinguish each class. We have confirmed that the images of the same class have the smallest distances to each other, and this point can be used in the cache.

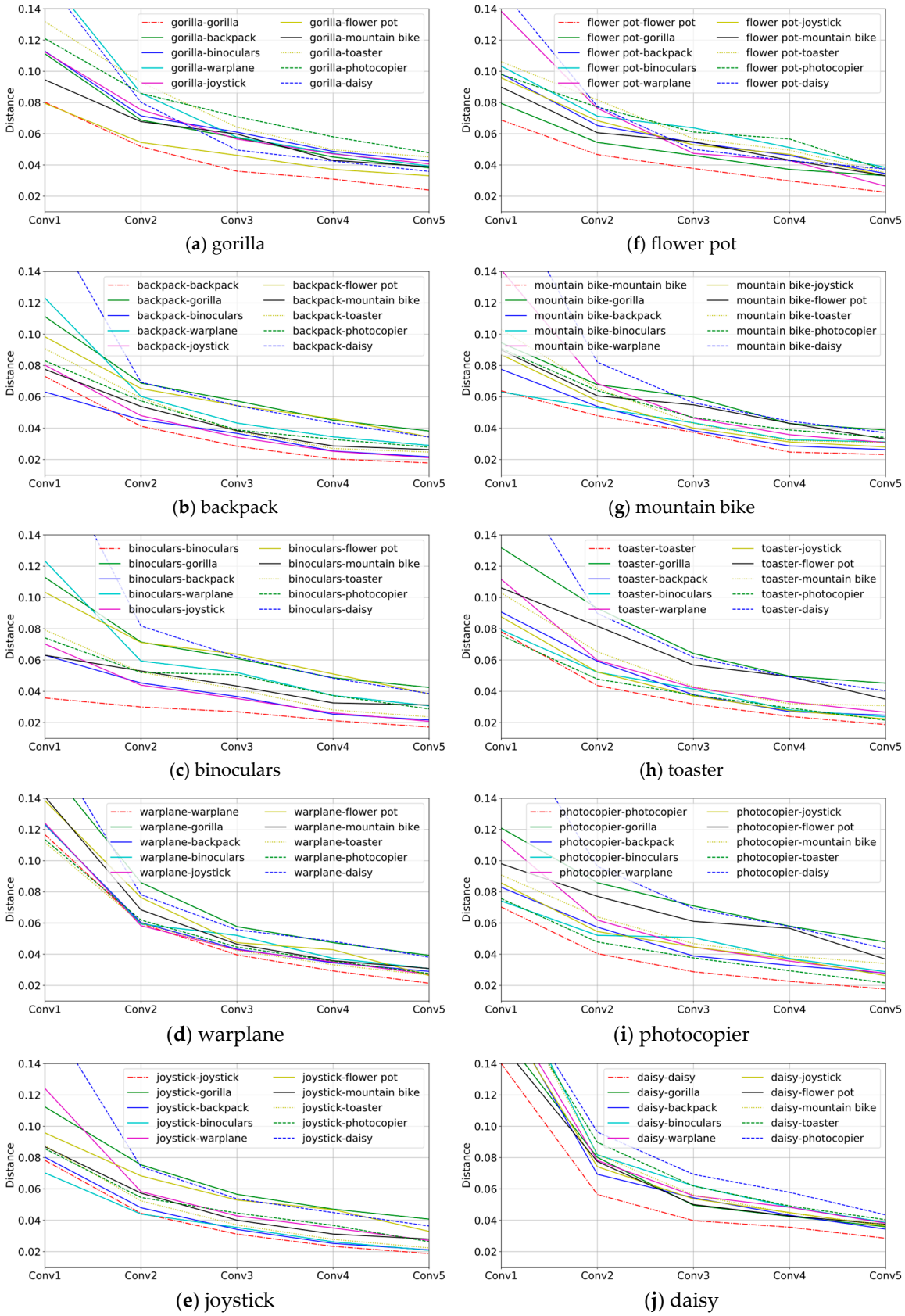


Figure 4. Distance between feature vectors for each class. The red dashed-dotted line is the distance between images of same class. The distances between images of the same class are less than those between images of different classes.

4.2. Cached CNN Hit Rate

To measure the hit rate of the proposed cached CNN, we selected 10 classes from the Microsoft COCO [22] dataset and selected 100 images for each class. This was done for a test and training set.

We first wrote training images in the cache of each convolution layer. We then classified 100 test images using a cache, and compared the results with the ground truth to measure the hit rate. We also experimented by increasing the number of cached training images from 2 to 10, 20, 40, 60, 80, and 100.

This experiment was divided into two stages. First, all test images were classified by the cache. In this case, we did not apply the distance threshold for *Hit*/*Miss* decision. That is, all results were a *Cache Hit*, and the query feature vector was only classified as the closest cached feature vector. In addition, the minimum number of cached images was one.

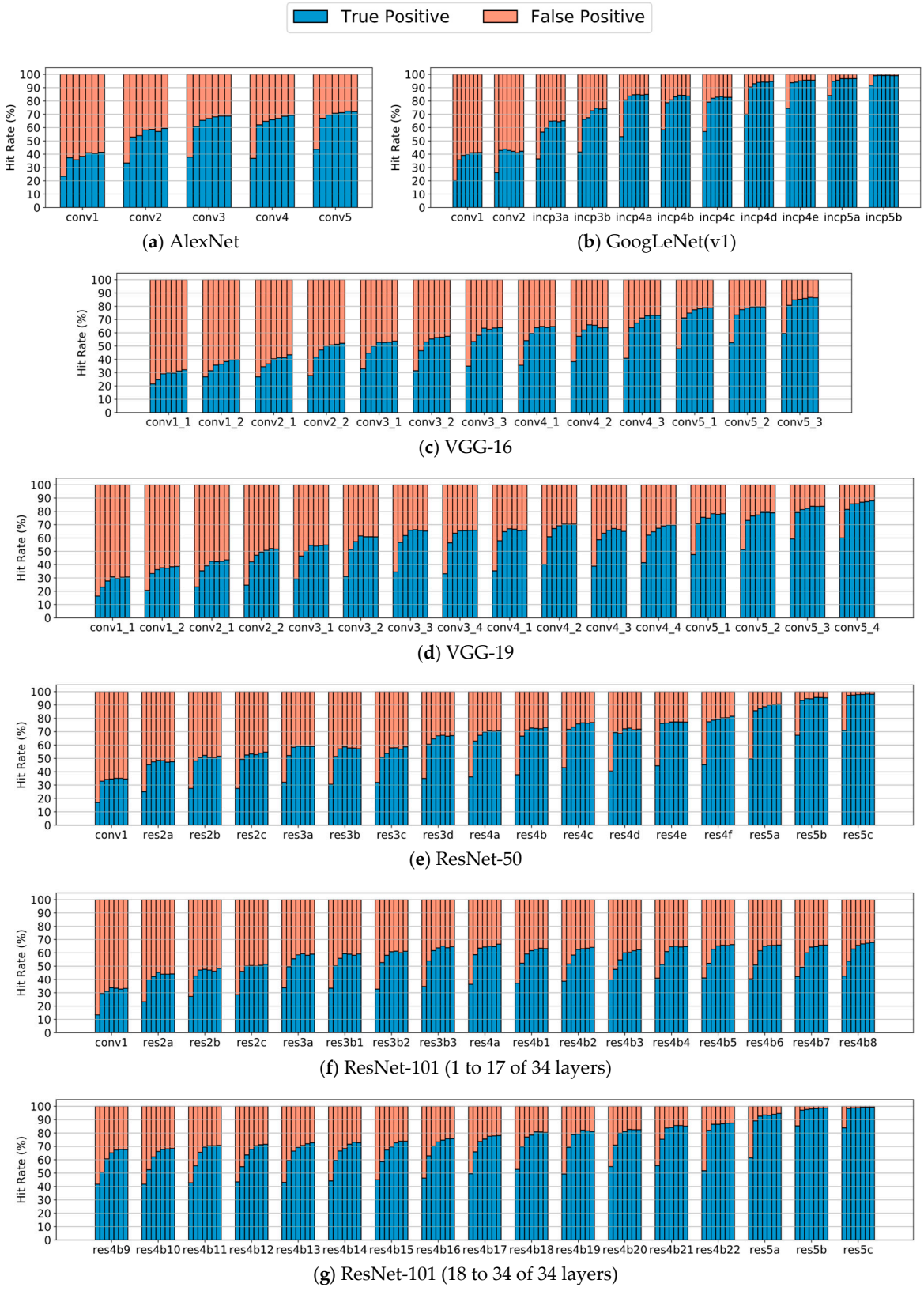
The second stage was the result of applying the distance threshold. Using these two experiments, we could decide how far the distance threshold can reduce false positives. The entire process of dataset selection, caching, and hit-rate measurement was performed on AlexNet, GoogLeNet [23], VGG-16, VGG-19 [24], ResNet-50, and ResNet-101 CNN models.

For example, the 10 selected classes and 100 images for each class were images classified as the same class by each CNN model from the Microsoft COCO dataset. In addition, GoogLeNet was measured in inception units, and ResNet was measured in residual block units.

Figure 5 shows the hit rate of classifying the query feature vector into a class of feature vectors that correspond to the closest distance without applying the distance threshold. In this case, the hit rate is 100 % because all input images are classified by the cache. Each item on the horizontal axis represents a convolution layer, which consists of seven bars. Each bar represents the number of cumulative cached training images of 1, 10, 20, 40, 60, 80, and 100. The blue and red areas of the bar represent the rate of true and false positives, respectively.

As expected, the layers toward the end extract more detailed and common features, so the rate of true positives is high. In addition, as the number of cached images increases, the rate of true positives increases, but from 40 onwards, it does not increase significantly, and in some cases the rate of true positive declines slightly. However, the rate of false positives is high except for the final layers of GoogLeNet, ResNet-50, and ResNet-101. In addition, given that all results are *Cache Hits*, the input query image may be classified as incorrect. Although CNNs, except AlexNet, show a true positive rate of 80 % or higher for the last convolution layer, it is insignificant to use the cache on this layer. Therefore, if the cache returns a false positive, it is advantageous to classify it using the CNN instead of using the cache, because it is necessary to lower the false positive rate and treat it as a *Cache Miss* even if there is a reduction in the hit rate of the cache.

Figure 6 shows the hit rate when applying the distance threshold to prevent a false positive from being a *Cache Miss* and to avoid using the cache. The seven bars for each item on the horizontal axis indicate that the number of training images accumulated in the cache is 2, 10, 20, 40, 60, 80, and 100.



(**a**) AlexNet (**b**) GoogLeNet(v1)

(**c**) VGG-16

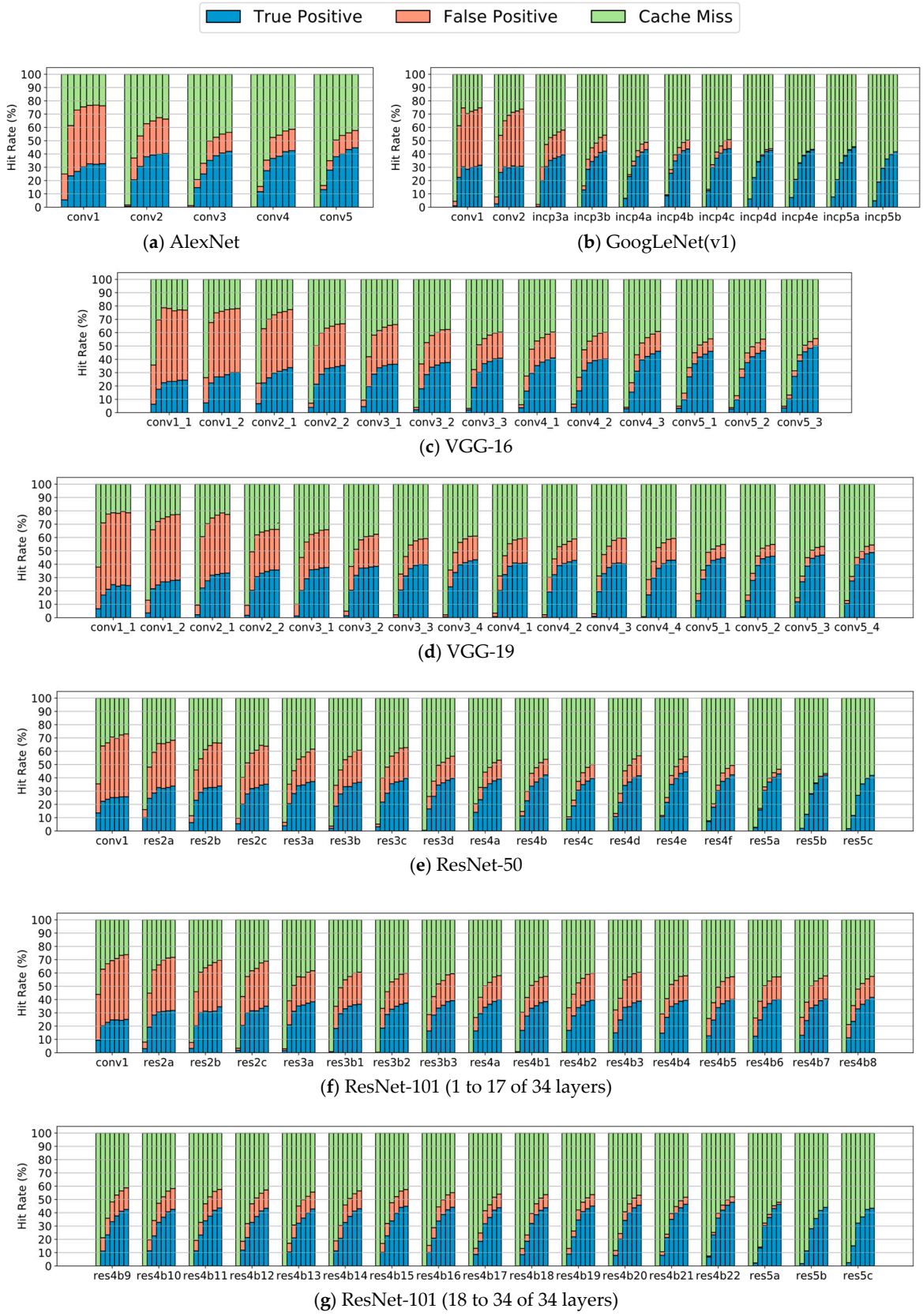
(**d**) VGG-19

(**e**) ResNet-50

(**f**) ResNet-101 (1 to 17 of 34 layers)

(**g**) ResNet-101 (18 to 34 of 34 layers)

Figure 5. Hit rate of cache without distance threshold. In this case, false positives are also classified by the cache, which are actually *Cache Misses*.



(**a**) AlexNet

(**b**) GoogLeNet(v1)

(**c**) VGG-16

(**d**) VGG-19

(**e**) ResNet-50

(**f**) ResNet-101 (1 to 17 of 34 layers)

(**g**) ResNet-101 (18 to 34 of 34 layers)

Figure 6. Hit rate of cache when applying the distance threshold. In this case, most false positives can be treated as *Cache Misses* to improve cache efficiency.

The green area of each bar is treated as a *Cache Miss* by applying the distance threshold. In this case, the input image can be classified by the CNN rather than the cache, the classification as a false positive can be avoided, and the hit rate is reduced by the *Cache Miss*.

By applying a distance threshold, the number of values retrieved from the cache, that is, the hit rate, has been greatly reduced. However, this also reduces the number of false positives, which in turn increases the usability of the cache. Table 1 lists the decreased amounts of true and false positives by applying the distance threshold according to each convolution layer. This value is the average of the results for all 10, 20, 40, 60, 80, and 100 cached training images.

Table 1. Decreased amounts of true and false positives when applying the distance threshold.

Models	Decreased Amounts (*percentage points*)																
AlexNet	9.4	21.9	33.7	33.2	36.0												
	17.3	**19.4**	**21.7**	**21.2**	**19.2**												
GoogLeNet (v1)	10.8	12.8	29.4	38.7	53.3	50.0	47.7	62.5	64.3	64.9	70.8						
	18.1	**19.6**	**20.9**	**19.3**	**12.6**	**13.2**	**13.4**	**5.9**	**4.7**	**3.5**	**0.8**						
VGG-16	6.9	9.5	10.4	17.7	19.5	22.4	26.7	28.7	29.1	33.9	42.4	43.7	48.2				
	17.0	**15.3**	**17.1**	**20.4**	**20.9**	**22.3**	**20.5**	**20.6**	**20.2**	**17.1**	**15.5**	**15.1**	**10.5**				
VGG-19	6.2	11.0	10.8	17.1	19.6	25.1	29.1	26.5	29.0	32.2	29.2	32.1	40.6	41.6	46.6	49.5	
	16.6	**15.3**	**16.1**	**20.8**	**20.6**	**19.6**	**19.5**	**19.9**	**19.3**	**18.1**	**19.3**	**18.4**	**15.7**	**15.6**	**12.2**	**9.4**	
ResNet-50	9.7	16.6	20.0	22.4	25.9	25.7	22.7	33.6	38.1	40.3	46.8	40.2	44.5	50.5	60.6	68.2	71.5
	20.9	**21.1**	**20.2**	**21.1**	**22.2**	**22.5**	**22.6**	**20.4**	**20.2**	**19.5**	**17.7**	**18.7**	**16.0**	**15.7**	**8.8**	**4.5**	**2.2**
ResNet-101 (1–17)	11.3	19.3	22.0	25.1	29.7	31.6	33.8	36.8	38.0	35.2	35.0	33.5	36.6	38.5	37.8	37.4	40.1
	24.2	**27.0**	**26.0**	**25.6**	**25.8**	**25.6**	**25.0**	**23.9**	**23.9**	**25.9**	**24.0**	**25.8**	**24.6**	**24.4**	**24.5**	**25.2**	**24.6**
ResNet-101 (18–34)	38.8	40.2	42.4	42.5	45.2	44.8	44.2	47.9	51.9	54.6	53.8	55.6	58.6	60.6	71.9	78.3	77.4
	25.4	**24.9**	**23.1**	**23.6**	**22.3**	**21.8**	**21.6**	**20.2**	**18.5**	**16.4**	**16.0**	**15.3**	**13.5**	**11.5**	**6.0**	**1.9**	**1.0**

The first row of each model shows the decreased amounts of true positives, while the second row in bold represents the decreased amounts of false positives. Depending on the CNN model, there is usually a decreased number of false positives in the initial layers. This is because these layers are filtered by the kernel to extract fewer common features that are filtered by the distance threshold.

In addition, the smaller amount of decrease in the final layers is a result of the fact that the number of false positives was already low even though these layers did not apply the distance threshold. However, the amount of true positive reduction is high because they were filtered by the distance threshold.

In the case of VGG and ResNet, the amount of decline of true positives was very high in the last layer. This is because the feature vectors corresponding to the true and false positives are scattered from the centroid of each class, and it is difficult to determine the boundary between the true and false positives of the feature vector only by the distance threshold.

Table 2 lists the hit rates after applying the distance threshold. This rate is the average of the measured values when the number of training images accumulated in the cache is 10, 20, 40, 60, 80, and 100. Before applying a distance threshold, all input images are classified by the cache, so the hit rate is 100 %, but the hit rate is reduced by applying a distance threshold and treating a portion of the false positive as a *Cache Miss*. The hit rate gradually decreases toward the final layer. This is because the number of false positives returned by the cache is very high in the initial layers; however, this decreases in the final layers.

Table 2. Hit rate when applying distance threshold.

Models	Hit Rate (%)																
AlexNet	73.3	58.7	44.5	45.6	44.9												
GoogLeNet (v1)	71.1	67.6	49.8	42.0	34.1	36.8	39.0	31.6	31.0	31.6	28.4						
VGG-16	76.2	75.3	72.5	61.9	59.6	55.3	52.9	50.8	50.8	49.1	42.1	41.2	41.3				
VGG-19	77.3	73.7	73.1	62.1	59.8	55.4	51.4	53.7	51.7	49.7	51.5	49.6	43.8	42.9	41.3	41.1	
ResNet-50	69.4	62.3	59.8	56.5	51.9	51.8	54.7	45.9	41.7	40.2	35.5	41.2	39.5	33.8	30.6	27.2	26.3
ResNet-101 (1–17)	64.5	53.7	52.0	49.3	44.6	42.8	41.3	39.3	38.1	39.0	41.0	40.8	38.9	37.2	37.7	37.5	35.4
ResNet-101 (18–34)	35.9	34.9	34.5	33.9	32.5	33.4	34.2	32.0	29.6	29.0	30.2	29.1	28.0	28.0	22.2	19.8	21.6

In addition, we measured the precision with and without the distance threshold. Table 3 shows this precision, which is measured as follows:

$$Precision = True\ Positive/(True\ Positive + False\ Positive) \tag{10}$$

Table 3. Precision of cache with and without distance threshold.

Models	Precision (*0 to 1*)																
AlexNet	0.39	0.57	0.67	0.66	0.71												
	0.41	**0.59**	**0.73**	**0.73**	**0.78**												
GoogLeNet (v1)	0.40	0.43	0.63	0.72	0.84	0.83	0.82	0.94	0.95	0.96	0.99						
	0.41	**0.44**	**0.67**	**0.79**	**0.91**	**0.89**	**0.89**	**0.99**	**0.99**	**1.00**	**1.00**						
VGG-16	0.30	0.37	0.40	0.49	0.51	0.54	0.61	0.62	0.63	0.70	0.77	0.78	0.85				
	0.30	**0.36**	**0.40**	**0.50**	**0.53**	**0.57**	**0.64**	**0.65**	**0.67**	**0.74**	**0.79**	**0.82**	**0.88**				
VGG-19	0.29	0.37	0.41	0.49	0.52	0.59	0.64	0.64	0.65	0.68	0.65	0.67	0.76	0.78	0.82	0.86	
	0.29	**0.35**	**0.41**	**0.51**	**0.54**	**0.61**	**0.67**	**0.69**	**0.69**	**0.72**	**0.68**	**0.70**	**0.80**	**0.83**	**0.86**	**0.88**	
ResNet-50	0.34	0.47	0.51	0.53	0.58	0.57	0.56	0.66	0.69	0.71	0.75	0.71	0.77	0.80	0.89	0.95	0.98
	0.36	**0.49**	**0.51**	**0.54**	**0.61**	**0.60**	**0.61**	**0.69**	**0.73**	**0.77**	**0.81**	**0.76**	**0.84**	**0.87**	**0.91**	**0.99**	**1.00**
ResNet-101 (1–17)	0.32	0.43	0.47	0.50	0.57	0.57	0.59	0.62	0.64	0.61	0.61	0.58	0.62	0.63	0.63	0.62	0.64
	0.34	**0.45**	**0.48**	**0.50**	**0.61**	**0.59**	**0.61**	**0.64**	**0.68**	**0.65**	**0.62**	**0.60**	**0.65**	**0.65**	**0.65**	**0.64**	**0.67**
ResNet-101 (18–34)	0.63	0.64	0.67	0.67	0.68	0.69	0.69	0.72	0.75	0.78	0.79	0.80	0.83	0.86	0.93	0.98	0.99
	0.67	**0.68**	**0.71**	**0.71**	**0.71**	**0.71**	**0.73**	**0.75**	**0.76**	**0.79**	**0.81**	**0.82**	**0.87**	**0.91**	**0.94**	**1.00**	**1.00**

This precision is the average of the results obtained when the number of training images accumulated in the cache is as in the case of the hit rate.

The first row of each model indicates the hit rate when no distance threshold is applied, while the second row shows the hit rate when a distance threshold is applied. Given that the number of true positives has decreased, the precision does not change significantly. Although the number of true positives retrieved is reduced owing to the application of the distance threshold, we focus on lowering the number of false positives. This is because the input query image is not classified using only the cache, and if it is classified as a false positive, it is more advantageous to classify it using the CNN without using the cache.

The precision increases at the last layer. However, achieving a high precision at this rear layer is insignificant. Thus, we measured the actual computation time and determined which layer is the most effective at using the cache.

4.3. Classification Time

We measured the classification time with the cache on two low-performance devices: ODROID [25] and LattePanda [26]. Table 4 shows their specifications.

Table 4. Specifications for devices used in experiment.

Models	Processor	Memory	Storage	Power
ODROID-XU4	Samsung Exynos5 Octa ARM Cortex-A15 Quad 2 GHz and Cortex-A7 Quad 1.3 GHz	2-GB DDR3	32GB MicroSD	5V/4A
LattePanda 4G/64GB	Intel Cherry Trail Z8350 Quad 1.8 GHz	4-GB DDR3	64GB eMMC	5V/2A

In these devices, the classification times required by a CNN without a cache, and with a cache at each convolution layer, were measured. Figure 7 shows the classification time required by the CNN and cache, and the hit rate of each device. The blue and green horizontal solid lines of each graph represent the classification times taken when one image was classified through CNN on each device. The blue and green dashed lines represent the classification times taken when classifying one image using the cache.

The measured classification time required by the CNN is the computation time from the first convolution layer to the last located layer that outputs the probability for each class, until the classified class is returned. The classification time required by the cache is the computation time from the first convolution layer to the return of the resultant class by the cache located in each layer. For example, in AlexNet, the classification time when using cache in conv3 includes the computation time of conv1 and conv2 layers.

The hit rate and precision are indicated by red solid and brown dashed-dotted lines, respectively. The hit rate includes true and false positives. All measured classification times were the average of the values obtained by repeating 100 times.

In AlexNet, caching at any layer could end the classification faster than the CNN. This means that the fully connected layer located behind the conv5 layer requires a large amount of computation.

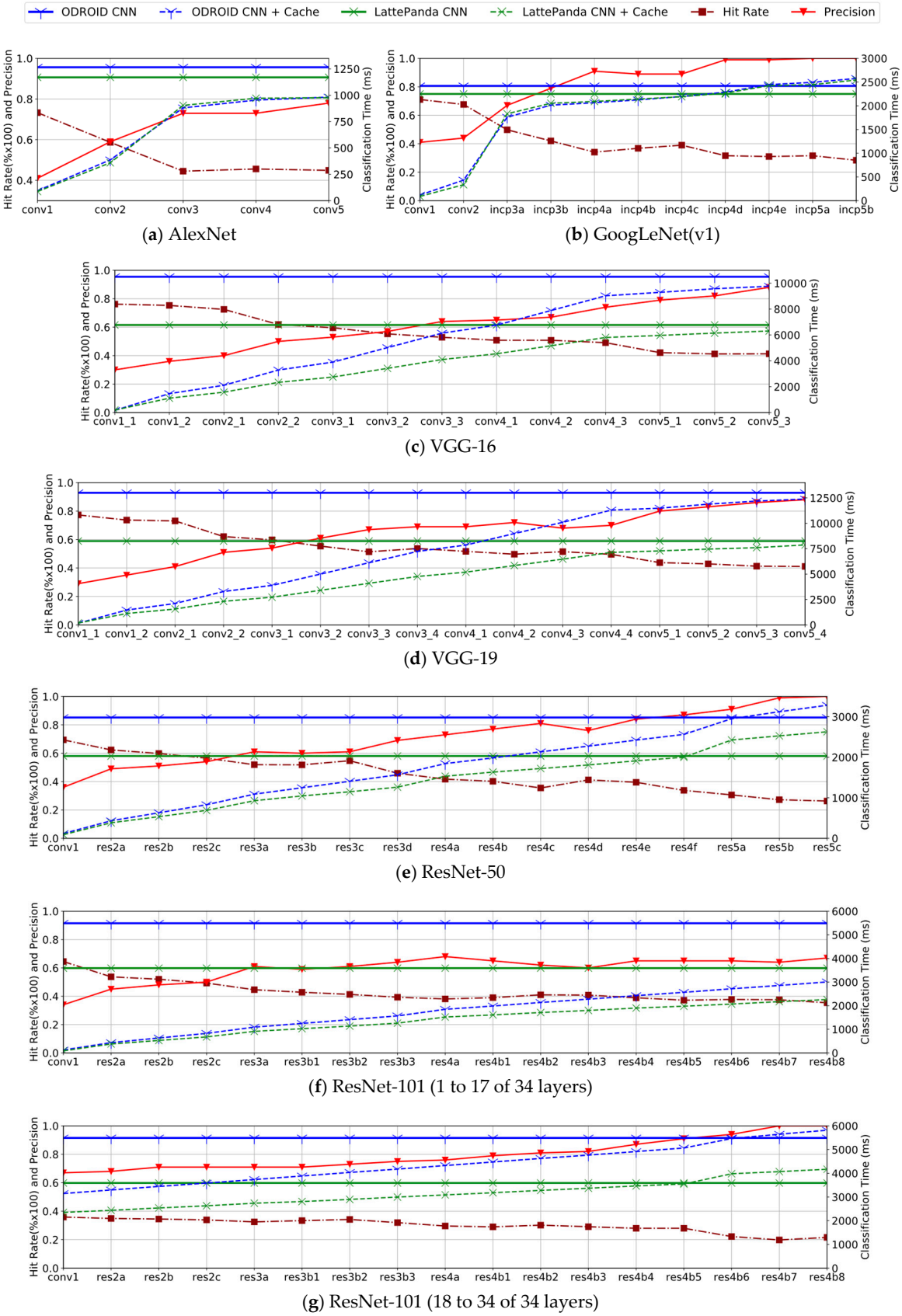
However, with our proposed cache, the classification time can be reduced because the classification result is obtained immediately after passing through the convolution layer without going through another layer such as the fully connected layer. However, the hit rate lowers from the conv3 layer. Nevertheless, as the precision increases inversely with the hit rate, it is advantageous to use the cache from conv3 onwards.

In GoogLeNet, ODROID, and LattePanda, the classification time with the cache increased over the CNN classification time in the inception4e and inception4d layer.

Therefore, the cache in GoogLeNet should be used in layers ahead of this layer. The hit rate gradually lowers toward the final layer; however, the precision increases rapidly. Hence, the best layer to place the cache can be selected based on a tradeoff with the classification time.

In VGG-16 and VGG-19, the classification time using the cache in all layers is less than that using the CNN. This allows the cache to be used even on the higher-precision final layers. Both the ResNet-50 and ResNet-101 classification times using the cache exceeded the CNN's classification time in the final layer. Thus, like GoogLeNet, the cache should be used in layers ahead of that layer.

As a result, the effect of classification time reduction depends on the precision. On an average, when the precision was approximately 0.7, 0.8, and 0.9, it took about 74.5, 85.0, and 93.9%, respectively, of the time to classify a single image as compared to when the cache was not used. This represents approximately 1.37, 1.18, and 1.06 times acceleration performance, respectively.



(**a**) AlexNet (**b**) GoogLeNet(v1)

(**c**) VGG-16

(**d**) VGG-19

(**e**) ResNet-50

(**f**) ResNet-101 (1 to 17 of 34 layers)

(**g**) ResNet-101 (18 to 34 of 34 layers)

Figure 7. Comparison of classification times required by the CNN and cache according to layer.

4.4. Result Based on Cache Location

Although the precision of the cache varies based on the network model, feature maps generated from the kernel are generally more precise on the final layers that extract features that are more common. However, when the cache is applied to the final layers, the classification time cannot be significantly reduced because the cache is operated after many convolution operations have already been performed. In addition, in GoogLeNet and ResNet, when the cache is located in the final convolution layers, the classification time increases, unlike when using the CNN. In this case, the cache is not available.

Experiments show that in most pretrained CNN models, the rate of false positives decreases gradually from the middle layers. As a low number of false positives indicates a high degree of precision, this cache can be used only if the number of false positives is low. The classification time owing to the cache also increases steadily from the first layer; hence, a tradeoff between the classification time and precision can be made. In addition, it may not be desirable to cache all layers or multiple layers. This is because the cache's computation time can exceed the CNN's classification time, as is the case with GoogLeNet and ResNet.

In our experiment, the classification time was measured by placing a cache in only one layer; however, in some cases, the classification time required by the cache exceeded the CNN's classification time. Table 5 shows the time spent searching the cache for each layer location. This time was measured when the number of classes written to the cache was 10. If the cache is located in only one layer, the search time of this cache does not take up a large portion of the total classification time. However, the more layers in which the cache is located, the more these values are added; hence, the classification time will increase. In addition, the resource and power consumption can also increase if the classification time is longer. Our proposed cached CNN can temporarily increase the resource and power consumption because it requires operations that extract feature vectors from feature maps and compare them to cached feature vectors. Moreover, if *Cache Misses* continue to occur, the resource and power consumption will increase significantly.

Table 5. Cache search time according to layer location measured by ODROID and LattePanda. Bold indicates LattePanda.

Models	Search Time (*ms*)																
AlexNet	22.8	49.9	63.9	63.9	43.2												
	35.5	**81.8**	**115.0**	**115.0**	**79.8**												
GoogLeNet (v1)	25.3	46.0	50.2	93.7	84.9	85.1	85.4	87.5	137.0	135.0	167.0						
	27.8	**65.3**	**82.4**	**153.0**	**152.0**	**152.0**	**152.0**	**157.0**	**245.0**	**245.0**	**304.0**						
VGG-16	62.6	62.3	48.3	48.3	60.1	59.5	59.9	97.9	97.4	97.9	83.4	83.4	83.1				
	52.1	**51.5**	**53.0**	**52.9**	**87.3**	**86.9**	**87.0**	**164.0**	**165.0**	**164.0**	**153.0**	**153.0**	**153.0**				
VGG-19	62.4	62.5	48.7	48.8	60.9	60.8	60.8	60.8	99.9	99.7	99.9	99.5	85.9	85.4	85.7	85.5	
	52.0	**53.6**	**52.8**	**53.0**	**86.4**	**88.3**	**86.3**	**86.4**	**163.0**	**166.0**	**162.0**	**166.0**	**153.0**	**153.0**	**153.0**	**153.0**	
ResNet-50	24.9	59.8	59.6	59.9	97.4	97.6	97.3	97.0	164.0	164.0	164.0	164.0	164.0	164.0	321.0	323.0	322.0
	27.9	**86.2**	**86.1**	**86.7**	**164.0**	**171.0**	**161.0**	**162.0**	**302.0**	**304.0**	**302.0**	**300.0**	**302.0**	**300.0**	**596.0**	**598.0**	**598.0**
ResNet-101 (1–17)	25.0	60.2	60.1	60.0	98.3	98.8	98.4	97.9	166.0	166.0	166.0	166.0	166.0	166.0	166.0	166.0	165.0
	27.7	**85.9**	**85.8**	**86.0**	**160.0**	**160.0**	**160.0**	**160.0**	**300.0**	**299.0**	**299.0**	**299.0**	**299.0**	**298.0**	**299.0**	**301.0**	**299.0**
ResNet-101 (18–34)	166.0	166.0	166.0	166.0	166.0	165.0	166.0	166.0	166.0	166.0	166.0	166.0	166.0	166.0	325.0	326.0	326.0
	299.0	**298.0**	**297.0**	**298.0**	**297.0**	**297.0**	**297.0**	**297.0**	**297.0**	**299.0**	**298.0**	**299.0**	**297.0**	**297.0**	**590.0**	**597.0**	**597.0**

However, if *Cache Hits* reduce the image classification time, the time required for the device to perform operations is also reduced; hence, the overall resource and power consumption may be reduced. In other words, if caches are executed from many convolution layers and *Cache Misses* continues to occur, not only does the classification time increase significantly, but also do the resources and power consumption. In this case, a tradeoff between the classification time and precision should be considered.

5. Conclusion and Future Work

We proposed a cache structure that can reduce the image classification time of a pretrained CNN. Previous studies modified CNN networks to reduce the classification time; however, the weights had to be retrained. The proposed structure can be used without modification to the structure or weight of the pretrained CNN. This cache is located in a specific layer of each CNN and stores the intensity of the feature maps that have occurred after the convolution operation on the input image at that layer. Subsequently, when a new image is input, the intensity of the feature maps from the convolution operation is compared to the cached intensity. If more than a certain degree of similarity is detected, it is possible to classify the cached feature maps into classes and terminate the remaining convolution layer operation. In other words, since the input image can be classified after performing only some convolution layer operations of the CNN, the classification time can be reduced.

This cache assists the CNN's image classification function. If the cache returns *Cache Miss* instead of *Cache Hit*, the remaining CNN operations can be continued to obtain the classification result of the input image. In addition, this cache is similar to some image classification systems, and we need to determine the precision of the returned value to be the ground truth.

Experiments have confirmed the effectiveness of the proposed method. This experiment was performed on two low-performance devices and we measured the classification time, hit rate, precision, and false positive rate when using the cache in six pretrained CNN models. The classification time was measured by comparing with and without the cache. From the measured classification time, the acceleration performance of the cache was confirmed. On an average, when the precision was about 0.7, 0.8, and 0.9, the classification speed was improved to 1.34, 1.18, and 1.06 times, respectively.

According to measurements, the false positive rate is still not low enough. Therefore, to make the proposed cache more useful, methods are needed to decrease the false positive rate. To achieve this, the cache of each layer may have a different distance threshold. However, an empirical approach is necessary because the feature maps are different for each network and for each layer. Otherwise, the cache may exist on multiple layers, in which case the output of each cache is voted on and the input image can be sorted into classes with the highest score.

Another method is to reduce the computation time of the cache. Our feature extraction and distance calculation methods are affected by the length of the feature vector, i.e., the number of kernels. The length of the feature vectors we use is twice the number of kernels; however, this length can be reduced by other feature vector extraction methods. If this length is reduced, the time spent in extracting and comparing feature vectors can be reduced. This can reduce the classification time of the cache, which can be used on higher-precision rear layers.

Our proposed cached CNN adds a cache operation; hence, if a *Cache Miss* occurs, it not only increases the classification time but also consumes resources and power temporarily. However, if a *Cache Hit* occurs, the computation can be terminated immediately, reducing resource and power consumption, and the classification time. Because resources and power are very important factors in low-performance devices, research is needed to find the optimum cache position.

Author Contributions: Writing–original draft, K.P.; Writing–review & editing, D.-H.K.

Acknowledgments: This work was supported by the Dual Use Technology Program (UM13018RD1).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105.
2. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.

3. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cuDNN: Efficient Primitives for Deep Learning. *arXiv*, 2014; arXiv:1410.0759v3.
4. Canziani, A.; Culurciello, E.; Paszke, A. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv*, 2016; arXiv:1605.07678v4.
5. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, Xi'an, China, 8–12 April 2017; pp. 615–629.
6. Teerapittayanon, S.; McDanel, B.; Kung, H.T. Branchynet: Fast Inference Via Early Exiting from Deep Neural Networks. In Proceedings of the 23rd International Conference on Pattern Recognition, Cancun, Mexico, 4–8 December 2016; pp. 2464–2469.
7. Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; Wierstra, D. Weight Uncertainty in Neural Networks. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1613–1622.
8. Collin, M.D.; Kohli, P. Memory Bounded Deep Convolution Networks. *arXiv*, 2014; arXiv:1412.1442v1.
9. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Network. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 1135–1143.
10. Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; Penksy, M. Sparse Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 806–814.
11. Xu, M.; Zhu, M.; Liu, Y.; Lin, F.X.; Liu, X. DeepCache: Principled Cache for Mobile Deep Vision. In Proceedings of the 24th International Conference on Mobile Computing and Networking, New Delhi, India, 29 October–2 November 2018; pp. 129–144.
12. Luan, S.; Chen, C.; Zhang, B.; Han, J.; Liu, J.I. Gabor Convolutional Networks. *IEEE Trans. Image Process.* **2018**, *27*, 4357–4366. [CrossRef] [PubMed]
13. Sundaram, A.; Ganesan, L.; Priyal, S.P. Texture Classification using Gabor wavelets based rotation invariant features. *Pattern Recogn. Lett.* **2006**, *27*, 1976–1982. [CrossRef]
14. Zhang, D.; Wong, A.; Indrawan, M.; Lu, G. Content-based Image Retrieval Using Gabor Texture Features. In Proceedings of the 1st IEEE Pacific-Rim Conference on Multimedia, Sydney, Australia, 13–15 December 2000; pp. 392–395.
15. Han, J.; Ma, K.-K. Rotation-invariant and Scale-invariant Gabor Features for Texture Image Retrieval. *Image Vis. Comput.* **2007**, *25*, 1474–1481. [CrossRef]
16. Sarwar, S.S.; Panda, P.; Roy, K. Gabor Filter Assisted Energy Efficient Fast Learning Convolutional Neural Networks. In Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design, Taipei, Taiwan, 24–26 July 2017; pp. 1–6.
17. Kinnikar, A.; Husain, M.; Meena, S.M. Face Recognition Using Gabor Filter and Convolutional Neural Network. In Proceedings of the International Conference on Informatics and Analysis, Pondicherry, India, 25–26 August 2016; pp. 1–4.
18. Budhi, G.S.; Adipranat, R.; Hartono, F.J. The Use of Gabor Filter And Back-Propagation Neural Network for the Automobile Types Recognition. In Proceedings of the 2nd International Conference on Soft Computing, Intelligent System and Information Technology, Bali, Indonesia, 1–2 July 2010.
19. Kwolek, B. Face Detection Using Convolutional Neural Networks and Gabor Filters. In Proceedings of the 15th International Conference on Artificial Neural Networks, Warsaw, Poland, 11–15 September 2005; pp. 551–556.
20. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Ong, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 675–678.
21. Griffin, G.; Holub, A.; Perona, P. *Caltech-256 Object Category Dataset*; California Institute of Technology: Pasadena, CA, USA, 2007.

22. Lin, T.-Y.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramana, D.; Zitnick, C.L.; Dollar, P. Microsoft COCO: Common Object in Context. In Proceedings of the 13th European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
23. 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 Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015.
24. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, 2014; arXiv:1409.1556v6.
25. Odroid-xu4:odroid-xu4. Available online: https://wiki.odroid.com/odroid-xu4/odroid-xu4 (accessed on 17 June 2018).
26. LattePanda 4 G/64 GB—LattePanda. Available online: https://www.lattepanda.com/products/3.html (accessed on 17 June 2018).