Article

Posture Recognition Using Ensemble Deep Models under Various Home Environments

Yeong-Hyeon Byeon ¹, Jae-Yeon Lee ², Do-Hyung Kim ² and Keun-Chang Kwak ^{1,*}

¹ Department of Control and Instrumentation Engineering, Chosun University, Gwangju 61452, Korea; qasdfghjt@hanmail.net

² Intelligent Robotics Research Division, Electronics Telecommunications Research Institute, Daejeon 61452, Korea; leeje@etri.re.kr (J.-Y.L.); dhkim008@etri.re.kr (D.-H.K.)

* Correspondence: kwak@chosun.ac.kr; Tel.: +82-062-230-6086

Received: 26 December 2019; Accepted: 11 February 2020; Published: 14 February 2020

Abstract: This paper is concerned with posture recognition using ensemble convolutional neural networks (CNNs) in home environments. With the increasing number of elderly people living alone at home, posture recognition is very important for helping elderly people cope with sudden danger. Traditionally, to recognize posture, it was necessary to obtain the coordinates of the body points, depth, frame information of video, and so on. In conventional machine learning, there is a limitation in recognizing posture directly using only an image. However, with advancements in the latest deep learning, it is possible to achieve good performance in posture recognition using only an image. Thus, we performed experiments based on VGGNet, ResNet, DenseNet, InceptionResNet, and Xception as pre-trained CNNs using five types of preprocessing. On the basis of these deep learning methods, we finally present the ensemble deep model combined by majority and average methods. The experiments were performed by a posture database constructed at the Electronics and Telecommunications Research Institute (ETRI), Korea. This database consists of 51,000 images with 10 postures from 51 home environments. The experimental results reveal that the ensemble system by InceptionResNetV2s with five types of preprocessing shows good performance in comparison to other combination methods and the pre-trained CNN itself.

Keywords: ensemble deep models; convolutional neural network; posture recognition; preconfigured CNNs; posture database; home environments

1. Introduction

Posture recognition is a technology that classifies and identifies the posture of a person and has received considerable attention in the field of computer vision. Posture is the arrangement of the body skeleton that arises naturally or mandatorily through the motion of a person. Posture recognition helps detect crimes, such as kidnapping or assault, using camera input [1] and also provides a service robot with important information for judging a situation to perform advanced functions in an automatic system. Posture recognition also helps rehabilitate posture correction in the medical field, is used as game content in the entertainment field, and provides suggestions to athletes for maximizing their ability in sports [2,3]. Additionally, it helps elderly people who live alone and have difficulty performing certain activities, by determining sudden danger from their posture in home environments. Thus, there are several studies on posture recognition because it is an important technique in our society.

Several studies have been performed on posture analysis, involving recognition, estimation, etc. Chan [4] proposed the scalable feedforward neural network-action mixture model to estimate three-dimensional (3D) human poses using viewpoint and shape feature histogram features extracted from a 3D point-cloud input. This model is based on mapping that converts a Bayesian network

to a feedforward neural network. Veges [5] performed 3D human pose estimation with Siamese equivariant embedding. Two-dimensional (2D) positions were detected, and then the detection was lifted into 3D coordinates. A rotation equivariant hidden representation was learned by the Siamese architecture to compensate for the lack of data. Stommel [6] proposed the spatiotemporal segmentation of keypoints given by a skeletonization of depth contours. Here, the Kinect generates both a color image and a depth map. After the depth map is filtered, a 2D skeletonization of the contour points is utilized as a keypoint detector. The human detection to a 2D clustering problem is simplified by the extracted keypoints. For all poses, the distances to other poses were calculated and arranged by similarity. Shum [7] proposed a method for reconstructing a valid movement from the deficient, noisy posture provided by Kinect. Kinect localizes the positions of the body parts of a person. However, when some body parts are occluded, the accuracy decreases, because Kinect uses a single depth camera. Thus, the measurements are objectively evaluated to obtain a reliability score for each body part. By fusing the reliability scores into a query of the motion database, kinematically valid similar postures are obtained. Commonly posture recognition is studied using the inertial sensor in a smart phone or other wearable devices. Lee [8] performed automatic classification of squat posture using inertial sensors via deep learning. One correct and five wrong squat postures were defined and classified using inertial data from five inertial sensors attached on the lower body, random forest, and convolutional neural network long short-term memory. Chowdhury [9] studied detailed activity recognition with a smartphone using trees and support vector machines. The data from the accelerometer in the smartphone were used to recognize detailed activity, like sitting on a chair not simply sitting. Wu [10] studied yoga posture recognition with wearable inertial sensors based on a two-stage classifier. The backpropagation artificial neural network and fuzzy C-means were used to divide yoga postures. Idris [11] studied human posture recognition using an android smartphone and artificial neural network. The gyroscope data from two smartphones attached to the arm were used to classify four gestures. To acquire data from inertial sensors or smartphones, a sensor usually needs to attach to the body, which is inconvenient for elderly people at home.

The development of deep learning has greatly exceeded the performance of machine learning; thus, deep learning is actively studied. Various models and learning methods have been developed. The depth of deep learning networks has expanded from tens to hundreds [12], in contrast to conventional neural networks with a depth of two to three. Deep learning networks abstract data to a high level through a combination of nonlinear transforms. There are many deep learning methods, such as deep belief networks [13], deep Q networks [14], deep neural networks [15], recurrent neural networks [16], and convolutional neural networks (CNNs) [17]. CNNs are designed by integrating a feature extractor and a classifier into a network to automatically train them through data and exhibit the optimal performance for image processing [18]. There are many pre-trained deep models based on the CNN, such as VGGNet [19], ResNet [20], DenseNet [21], InceptionResNet [22], and Xception [23]. These pre-trained models can be used for transfer learning. Transfer learning reuses the parameters of the models, which are trained from a large-scale database using high-performance processing units for a long time. Transfer learning can be effectively used in cases where a neural network needs to be trained with a small database and there is insufficient time to train the network with new data [24].

There are many studies on posture estimation based on deep learning. Thompson [25] used a CNN to extract several scale features of body parts. These features include a high-order spatial model sourced from a Markov random field and indicate the structural constraint of the domain between joints. Careira [26] proposed a feedback algorithm employing a top-down strategy called iterative error feedback. It carries the learning hierarchical representation of the CNN from the input to the output with a self-modifying model. Pishchulin [27] concentrated on local information by reducing the receptive field size of the fast R-CNN (Region based CNN) [28]. Thus, partial detection was converted into multi-label classification and combined with DeepCut to perform bottom-up inference. Insafutdinov [29] proposed residual learning [20] that includes more context by increasing the size of the receptive field and the depth of the network. Georgakopoulos [30] proposed a methodology

for classifying poses from binary human silhouettes using a CNN, and the method was improved by image features based on modified Zernike moments for fisheye images. The training set is composed of a composite image created from a 3D human model using the calibration model of the fisheye camera. The test set is the actual image acquired by the fisheye camera [31]. To increase performance, many studies have used ensemble deep models at various applications. Lee [32] designed an ensemble stacked auto-encoder based on sum and product for classifying horse gaits using wavelet packets from motion data of the rider. Maguolo [33] studied an ensemble of convolutional neural networks trained with different activation functions using sum rule to improve the performance in small- or medium-sized biomedical datasets. Kim [34] studied deep learning based on 1D ensemble networks using an electrocardiogram for user recognition. The ensemble network is composed of three CNN models with different parameters and their outputs are combined into single data.

Traditionally, to recognize posture, it was necessary to obtain the coordinates of the body points or inertial data. This was achieved using a depth camera such as Kinect, image processing through a body model, or devices for capturing motion connected to the body; regarding the latter, it is a nuisance to wear these sensors with care in everyday life. Posture recognition, using images that do not require a sensor attached to the body, does not have this problem. Since posture recognition is performed using images, it can be applied to inexpensive cameras, and the device used for acquiring experimental data also has an inexpensive feature even though it supports a depth camera. In conventional machine learning, there is a limitation to recognizing posture directly using only an image [12,35–37]. However, owing to advancements in deep learning, good performance in posture recognition can be achieved using only one image. In the present study, several pre-trained CNNs were employed for recognizing the posture of a person using various preprocessed 2D images. To train the deep neural network for posture recognition, a large number of posture images was required. Therefore, a daily domestic posture database was directly constructed for posture recognition under the assumption of an environment of domestic service robots. The database includes ten postures: “standing normal”, “standing bent”, “sitting sofa”, “sitting chair”, “sitting floor”, “sitting squat”, “lying face”, “lying back”, “lying side”, and “lying crouched”. The training, validation, and test sets were real captured images, not synthetic images. Moreover, ensemble CNN models were studied to improve performance. The performances of posture recognition using ensemble CNNs with various types of preprocessing, not studied thus far, were compared. In the case of single models, type 2 exhibited 13.63%, 3.12%, 2.79%, and 0.76% higher accuracy than types 0, 1, 3, and 4, respectively, under transfer learning; and VGG19 exhibited 15.78%, 0.78%, 3.53%, 4.11%, 10.61%, and 16.02% higher accuracy than the simple CNN, VGG16 [19], ResNet50 [20], DenseNet201 [21], InceptionResNetV2 [22], and Xception [23], respectively, under transfer learning. In the case of ensemble systems, the ensemble system combining InceptionResNetV2s using average of scores was 2.02% higher than the best single model in our experiments under nontransfer learning. We performed posture recognition that can be applied to general security cameras to detect and respond to human risks. To do so, we acquired a large amount of data for training a neural network. In order to use the existing CNN models with fixed input form, various methods were applied to fit the input form. We compared the performance of posture recognition by applying various existing CNN models, and proposed ensemble models by input forms and CNN models.

We performed posture recognition using ensemble preconfigured deep models in home environments. Section 2 describes the deep models of the CNN. Section 3 discusses the database and experimental methods. Section 4 presents the experimental results, and Section 5 concludes the paper.

2. Deep Models of CNN

There are many pre-trained deep models based on the CNN, such as VGGNet [19], ResNet [20], DenseNet [21], InceptionResNet [22], and Xception [23]. These pre-trained models can be used for transfer learning. Transfer learning reuses the parameters of the models, which are trained from a large-scale database using high-performance processing units for a long time. Transfer learning can

be effectively used in cases where a neural network needs to be trained with a small database and there is insufficient time to train the network with new data [24].

2.1. VGGNet

VGGNet is ranked second in the ILSVRC 2014, following GoogLeNet, but it is more widely used than GoogLeNet because it has a significantly simpler structure. VGGNet uses a relatively small 3×3 convolution (conv) filter and a 1×1 convolution filter, in contrast to AlexNet's 11×11 filters in the first layer and ZFNet's 7×7 filters. When the nonlinear rectified linear unit (ReLU) activation function is applied after the 1×1 convolution, the model becomes more discriminative. Additionally, a smaller filter requires less parameters to be learned and results in higher processing speed. In VGGNet, a maxpool with a 2×2 kernel size and 2 strides, 2 fully connected (FC) layers with 4096 nodes, 1 FC layer with 1000 nodes, and 1 Softmax layer is used. Two 3×3 convolution layers and three 3×3 convolution layers have effective receptive fields of 5×5 and 7×7 , respectively. By doubling the number of filters after each maxpool layer, the spatial dimensions are reduced, but the network depth is increased. Originally, VGGNet was designed to investigate how errors are affected by the network depth. There are models with layer depths of 8, 11, 13, 16, and 19 in VGGNet. As the network depth increases, the error decreases, but the error increases if the layer depth exceeds 19. VGGNet uses data augmentation of scale jittering for training and is trained with batch gradient descent using four Nvidia Titan Black graphics processing units for approximately three weeks [19]. Figure 1 shows the structure of VGGNet. In an expression with the form $L@M \times N$, L , M , and N represent the sizes of the map, the row of the kernel, and the column of the kernel, respectively. The @ is a symbol used to separate the map size and filter size here.

Figure 1. Structure of VGGNet. FC, fully connected.

2.2. ResNet

Deep layers of neural networks result in a vanishing gradient, an exploding gradient, and degradation. The vanishing gradient refers to when a propagated gradient becomes too small, and exploding gradient refers to when a propagated gradient becomes too large to train. Degradation indicates that the deep neural network has worse performance than the shallow neural network even though there is no overfitting. ResNet attempts to solve these problems by reusing the input features of the previous layer. Figure 2 shows the structure of ResNet. The output of Y is calculated using the input of X , and the input of X is reused by being added to the output of Y . This is called a skip connection. Then, learning is performed so that $\text{ReLU}(W \times X)$ converges to 0, indicating that the output of Y is almost equal to X . This reduces the vanishing gradient, and even small changes in the input are delivered to the output. The number of intermediate layers in the section of the skip connection can be set arbitrarily, and ResNet uses this method to stack layers deeply. ResNet is structured according to VGGNet [19] and uses the convolution of 3×3 filters,

but not pooling or dropout. The pooling is replaced with the convolution of 2 strides. After every two convolutions, the input layer is added at the output [20].

Figure 2. Structure of ResNet.

2.3. DenseNet

A typical network structure is a sequential combination of convolution, activation, and pooling. In contrast to the typical network, DenseNet solves the degradation problem by introducing a new concept called dense connectivity. DenseNet has approximately 12 filters per layer and uses the dense connectivity pattern to continuously pile up feature maps in previous layers, effectively conveying the information in the early layers to the latter layers. This allows entire feature maps to enter within the network evenly into the last classifier, while simultaneously reducing the total number of parameters, making the network sufficiently learnable. The dense connection also functions as regularization, which reduces the overfitting even for small datasets. Dense connectivity is expressed by Equation (1) and shown in Figure 3. DenseNet divides the entire network into several dense blocks and groups layers with the same size of the feature map into the same dense block. The part of pooling and convolution is called the transition layer. This layer comprises a batch normalization (BN) layer, a 1×1 convolution layer for adjusting the dimensions of the feature map, and a 2×2 average pooling layer. A bottleneck structure (i.e., BN-ReLU-conv(1)-BN-ReLU-conv(3)), is employed to reduce the computational complexity. Usually, global average pooling is used instead of FC, which most networks have in the last layer. The network is trained via stochastic gradient descent [21]. Figure 4 shows the structure of DenseNet.

$$x_m = H_m([x_0, x_1, \dots, x_m]) \quad (1)$$

Figure 3. Dense connectivity.

Figure 4. Structure of DenseNet.

2.4. InceptionResNetV2

The objective of the inception module is to cover a wide area of the image while maintaining the resolution for smaller information. Thus, convolutions of different sizes from 1×1 to 5×5 are performed in parallel. The inception module first performs convolution with a 1×1 filter and then performs convolutions with filters of different sizes, because the convolution with a 1×1 filter reduces the number of feature maps and thus the computational cost. The results of convolutions performed in parallel are concatenated at the output layer of the inception module. The InceptionResNetV2 consists of a stem layer, three types of inception modules (A, B, and C), and two types of reduction modules (A and B). The stem layer is the frontal layer, and the reduction module is used to reduce the size of the feature map in InceptionResNetV2. The inception modules of InceptionResNetV2 are based on the integration of the inception module and the skip connection of ResNet [22]. The stem layer, three types of inception modules (A, B, and C), two types of reduction modules (A and B), and the structure of InceptionResNetV2 are shown in Figures 5–8, respectively. In an expression with the form $L \times M \times N : O$, L , M , N , and O represent the sizes of the map, the row of the kernel, the column of the kernel, and the stride, respectively.

Figure 5. Stem layer of InceptionResNetV2. BN, batch normalization; ReLU, rectified linear unit.

2.5. Xception

Xception, which is based on inception, seeks to completely separate the search for relationships between channels from the search for regional information on images. In Xception, depth-wise separable convolution is performed for each channel, and the result is projected to the new channel space via 1×1 convolution. If the existing convolution created a feature map considering all the channels and local information, the depth-wise convolution creates one feature map for each channel, and then 1×1 convolution is performed to adjust the number of feature maps. The 1×1 convolution is called the pointwise convolution (point-conv). In inception, each convolution is followed by the nonlinearity of the ReLU; however, in a depth-wise separable convolution, the first convolution is not followed by nonlinearity. Xception has 36 convolution layers in 14 modules for feature extraction. Except for

the beginning and end layers, each module has a linear residual connection. In summary, Xception is formed by linearly stacking depth-wise separable convolution layers with residual connections [23]. Figure 6 shows the structure of Xception.

Figure 6. Structure of Xception.

3. Database and Experiment Method

3.1. Construction of Database for Posture Recognition

Training a deep neural network for posture recognition requires a large amount of posture images. Thus, the daily domestic posture database was constructed for posture recognition under the assumption of an environment of domestic service robots. Astra (i.e., an assembly of sensors for color images (red, green, and blue (RGB)), depth images (infrared rays), and sound sources (microphones) made by Orbbec), was used to construct a daily domestic posture database. It senses the depth from 0.6 to 8 m and the color inside a field of view of $60^\circ \times 49.5^\circ \times 73^\circ$. The resolution and frame rate are 640×480 pixels and 30 fps, respectively, for the RGB and depth images. The size of Astra is $165 \times 30 \times 40$ m³, its power consumption is 2.4 W (through Universal Serial Bus (USB) 2.0), and it has two microphones. Figure 7 shows the Astra for capturing posture images. It can be developed using the Astra software development kit (SDK) and OpenNI in several operating systems (e.g., Android, Linux, and Windows 7, 8, and 10). The SDK also supports body tracking with a skeleton [38]. To construct

the database, a graphical user interface (GUI) for the capture tool was developed using the Microsoft Foundation Class library and the Astra SDK in Windows 7. Figure 8 shows the developed GUI for constructing the posture database.

Figure 7. Astra camera for capturing posture images. RGB, red, blue, green; USB, Universal Serial Bus.

Figure 8. Graphical user interface (GUI) for constructing the posture database.

A total of 51 homes participated in the construction of the daily domestic posture database. These homes had a living room, a kitchen, a small room, a dinner table, chairs, a bed, and a sofa. More than two persons in each home equally contributed as subjects for the posture images. Ten postures were defined to construct the daily domestic posture database. The postures were “standing normal”, “standing bent”, “sitting sofa”, “sitting chair”, “sitting floor”, “sitting squat”, “lying face”, “lying back”, “lying side”, and “lying crouched”. Each home generated 100 images per posture, and each image included one subject. Each home generated 1000 posture images (10 postures $\times$ 100 images per posture). Thus, the total number of images was 51,000 (1000 posture images $\times$ 51 homes). Each image was captured by varying the position, type of room, prop (such as clothes), furniture (such as a chair), pose of the sensor, pose of the person, and small movement. Figure 9 shows the environment for capturing the posture images using Astra. Figure 10 shows examples of posture images captured using Astra.

Figure 9. Environment for capturing posture images using Astra.

Figure 10. Cont.

Figure 10. Examples of posture images captured from Astra camera in various home environments: (a) standing normal; (b) standing bent; (c) sitting sofa; (d) sitting chair; (e) sitting floor; (f) sitting squat; (g) lying face; (h) lying back; (i) lying side; (j) lying crouched.

3.2. Preprocessing Types

To segment the person images, you only look once (YOLO)-v3 was used as a person detector [39]. After segmenting the person, preprocessing was performed to consider various input methods which have different pros and cons because the neural networks were limited in input as a square fixed image. The pros and cons are based on tightness of person and distortion of person by stretching image. Five types of preprocessing for cropping images were defined to input posture images into the neural network: In type 0, the original image was resized to the size of the input layer. This changed the original ratio of the image. In type 1, the person image was segmented from the original image while satisfying the size of the input layer. The original ratio of the image was maintained. In type 2, the person image was tightly segmented from the original image regardless of the size of the input layer. Then, the person image was resized while maintaining the ratio of the original image until the size of the row or column of the person was equal to the size of the row or column of the input layer, respectively. The extra border due to the difference in the image ratio between the person and the input layer was zero-padded. In type 3, the person image was tightly segmented from the original image regardless of the size of the input layer. Then, the person image was resized to the size of the input layer. This changed the original ratio of the image. In type 4, the person image was tightly segmented from the original image regardless of the size of the input layer. Then, the person image was resized while maintaining the ratio of the original image until the size of the row or column of the person was equal to the size of the row or column of the input layer, respectively. The extra border due to the difference in the image ratio between the person and the input layer was replicated to the edge of the original image. Figure 11 shows the examples of five types of original and preprocessed images.

Figure 11. Examples of five types of original and preprocessed images: (a) original image; (b) type 0; (c) type 1; (d) type 2; (e) type 4; (f) type 5.

3.3. Posture Recognition Using Ensemble CNN Models

As more people live alone, there is a need for ways to cope with crime or dying alone. Posture recognition emerged as a solution because it contains some information about the person's situation. However, posture recognition using the conventional inertial sensor, the sensor is cumbersome to attach to the body. For the elderly, it is difficult to wear the sensor all the time and extra help is needed for them to do so. However, image-based posture recognition through a camera can be applied to the existing camera system and there is no need to attach any sensors. In addition, low-cost webcams can also be applied because the posture is recognized only by the image. It has been difficult to recognize posture by using only two-dimensional images, but recently, deep learning has proved excellent in various fields. We applied the technique to the 2D image based posture recognition. First, a daily domestic posture database was constructed. Then, the posture images were processed to different types of preprocessing. The CNN models were VGG16, VGG19 [19], ResNet50 [20], DenseNet201 [21], InceptionResNetV2 [22], and Xception [23]. Transfer learning is effectively used in cases where a neural network needs to be trained with a small database and there is insufficient time to train the network with new data. In this study, posture recognition was performed by training the CNNs using transfer learning. Most of the parameters used to extract the feature maps were fixed, and only the final FC layers and Softmax were trained. In addition, CNNs were trained by updating all the parameters (nontransfer learning). Figure 12 shows the posture recognition using the preprocessed images for different CNN models. Then CNNs of the deep models and CNNs of the preprocessing types are combined using the majority of outputs and average of scores as ensemble methods. The majority of outputs decides the most voted class among outputs of CNNs. The average of score decides the class with a max score from averaged scores of CNNs. Tables 1 and 2 show the ensemble methods of majority vote and score average. Figure 13 shows the various ensemble systems by input types and CNN models.

Table 1. Ensemble method of majority vote.

1.	Net results scores of each class as $1 \times n$ matrix.
	$s_c = [x_1 \ x_2 \ x_3 \ \dots \ x_n]$
2.	Get the outputs by picking the class to the max value in scores as one hot matrix.
	$o_c = [0 \ 0 \ 1 \ \dots \ 0]$
3.	Sum the one hot matrix of all classifiers.
	$v = \sum_{c=1}^C o_c$
4.	Get the final output by picking the class to the max value in sum of one hot matrix.
	$o_v = [1 \ 0 \ 0 \ \dots \ 0]$

Table 2. Ensemble method of score average.

1.	Net results scores of each class as $1 \times n$ matrix.
	$s_c = [x_1 \ x_2 \ x_3 \ \dots \ x_n]$
2.	Calculate the average of scores of all classifiers.
	$a = \frac{1}{C} \sum_{c=1}^C s_c$
3.	Get the final output by picking the class to the max value in average of scores.
	$o_a = [1 \ 0 \ 0 \ \dots \ 0]$

The number of postures is the number of classes. The number of defined postures was 10: “standing normal”, “standing bent”, “sitting sofa”, “sitting chair”, “sitting floor”, “sitting squat”, “lying face”, “lying back”, “lying side”, and “lying crouched”. The 10 posture classes were divided into four categories: standing, sitting, lying, and lying crouched. Instead of 10 and 4 classes, there were 11 and 5 when other images, such as the background, were considered as a separate class. The results of posture recognition under 10, 4, 11, and 5 classes were obtained. The other images were configured by programmatically cropping the original posture images randomly while avoiding the person to the greatest extent possible. These various numbers of classes have the advantage to extract context information. For example, a person does not sleep in a standing posture. Figure 14 shows the four categories.

Figure 12. Ensemble deep models using the preprocessed images with different convolutional neural network (CNN) models.

Figure 13. Various ensemble deep models by input types and the pre-trained CNNs.

Figure 14. Examples of four categories with 10 postures.

The posture-recognition performance was evaluated with regard to accuracy, which was defined as the number of correct classifications (CC) divided by the number of total classifications (i.e., the sum of CC and the number of wrong classifications (WC)) [40]:

$$Accuracy = \frac{CC}{CC + WC} \quad (2)$$

4. Experimental Results

4.1. Database

The daily domestic posture database was directly constructed from 51 homes using Astra. All homes had a living room, a kitchen, a small room, a dinner table, chairs, a bed, and a sofa. The postures included “standing normal”, “standing bent”, “sitting sofa”, “sitting chair”, “sitting floor”, “sitting squat”, “lying face”, “lying back”, “lying side”, and “lying crouched”. Each home had 100 images per posture, and each image included one subject. Each home had 1000 posture images (10 postures × 100 images per posture). The total number of images was 51,000 (1000 posture images × 51 homes). The images were captured by varying the position, type of room, prop (such as clothes), furniture (such as a chair), pose of the sensor, pose of the person, and small movement. The posture database was constructed with men and women ranging from 19 to 68 years of age. Figure 15 shows the data configuration for the training, validation, and testing. The posture images were subjected to different types of preprocessing. To segment the person image, YOLO-v3 was used as a person detector [39], and the segmented person images were filtered manually. The number of preprocessed posture images for the training was 3910 for standing normal, 3802 for standing bent, 3899 for sitting sofa, 3899 for sitting chair, 3896 for sitting floor, 3859 for sitting squat, 3752 for lying face, 3777 for lying back, 3773 for lying side, 3505 for lying crouched, and 6802 for other images. The number of preprocessed posture images for the validation was 689 for standing normal, 670 for standing bent, 688 for sitting sofa, 687 for sitting chair, 687 for sitting floor, 681 for sitting squat, 662 for lying face, 666 for lying back, 665 for lying side, 618 for lying crouched, and 1200 for other images. The number of

preprocessed posture images for the testing was 500 for standing normal, 497 for standing bent, 500 for sitting sofa, 500 for sitting chair, 497 for sitting floor, 498 for sitting squat, 489 for lying face, 489 for lying back, 482 for lying side, 481 for lying crouched, and 2001 for other images. The total number of images for the training, validation, and testing was 44,874, 7913, and 6934, respectively.

Figure 15. Data configuration for training, validation, and testing data set.

4.2. Experimental Results

The computer used in the experiment had the following specifications: Intel i7-6850K central processing unit at 3.60 GHz, Nvidia GeForce GTX 1080 Ti, 64 GB of random-access memory, and Windows 10 64-bit operating system. The CNN models used for posture recognition in this study were VGG16, VGG19 [19], ResNet50 [20], DenseNet201 [21], InceptionResNetV2 [22], and Xception [23], which are pre-trained CNNs. A pre-trained CNN can be used for transfer learning when new data need to be classified efficiently. Most of the parameters used to extract the feature maps were fixed, and only the final FC layers and Softmax were trained. A simple CNN (the conventional method) was added to the experiment for performance comparison. The structure of the simple CNN is depicted in Figure 16. The simple CNN was trained by updating all the parameters, because it was not a pre-trained model. Here, the number of postures was the number of classes. The number of defined postures was 10: “standing normal”, “standing bent”, “sitting sofa”, “sitting chair”, “sitting floor”, “sitting squat”, “lying face”, “lying back”, “lying side”, and “lying crouched”. If a class of “other images” (e.g., background) was added to the 10 posture classes, then the number of total classes was 11. The training, validation, and testing data sets for the 11 classes consisted of 44,874, 7913, and 6934 images, respectively. Additionally, the 10 posture classes were divided into four categories: standing, sitting, lying, and lying crouched. If a class for “other images” was added to these four posture classes, the number of total classes was five. The performance for the other numbers of classes was measured through simple mapping of the trained model with the data of 11 classes. The simple CNN was trained with a batch size of 128, 50 epochs, and the RMSProp optimizer [41]. The other pre-trained models were trained via transfer learning with a batch size of 128, 30 epochs, and the Adadelta optimizer [42]. The highest accuracies without data augmentation under transfer learning were 78.35% for VGG19 and type 2 preprocessing with the posture data of 11 classes, 69.89% for VGG19 and type 2 preprocessing with the posture data of 10 classes, 88.86% for VGG19 and type 2 preprocessing with the posture data of five classes, and 88.50% for ResNet50 and type 1 preprocessing with the posture data of four classes. Table 3 presents the classification performance of the CNNs without data augmentation under transfer learning. Figure 17 shows the confusion matrix for VGG19 with type 2 preprocessing, which exhibited the best performance for the posture data of 11 classes without data augmentation under transfer learning. Tables 4–6 present the accuracies of the CNNs for the posture data of 10, 5, and 4 classes, respectively, without data augmentation under transfer learning.

Figure 16. Structure of the simple CNN.

Table 3. Classification performance of the CNNs without data augmentation under transfer learning (11 classes).

Single Model (%)		Type 0	Type 1	Type 2	Type 3	Type 4
SimpleCNN	Training	91.50	70.82	81.12	85.42	68.87
	Validation	75.65	58.70	63.74	57.98	59.34
	Testing	45.80	61.03	67.18	60.85	63.48
VGG16	Training	89.40	88.44	91.72	85.87	85.76
	Validation	74.50	74.62	76.92	72.65	74.59
	Testing	63.44	70.12	76.64	70.78	75.31
VGG19	Training	90.09	91.17	89.56	83.06	86.18
	Validation	74.51	74.22	77.18	73.12	74.93
	Testing	64.87	69.82	78.35	71.46	74.94
ResNet50	Training	60.03	71.56	60.67	67.48	68.05
	Validation	58.44	69.10	59.92	65.61	65.64
	Testing	55.60	66.83	57.92	65.55	61.81
DenseNet201	Train	62.17	70.52	62.53	70.65	59.44
	Validation	60.68	67.56	61.27	68.44	58.34
	Test	54.51	64.16	55.88	66.51	50.68
InceptionResNetV2	Train	44.98	55.63	55.10	59.68	56.01
	Validation	43.86	53.67	54.71	59.39	55.90
	Test	40.28	49.35	53.20	59.61	52.48
Xception	Train	43.56	57.12	51.18	57.21	56.52
	Validation	42.34	54.73	49.62	55.98	54.15
	Test	34.74	50.03	46.09	54.66	52.32

Figure 17. Confusion matrix for VGG19 with preprocessing of type 2 without data augmentation.

Table 4. Classification performance of the CNNs without data augmentation under transfer learning (10 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
SimpleCNN	25.36	48.16	53.88	51.63	51.54
VGG16	48.95	62.63	67.34	63.65	66.79
VGG19	51.18	61.94	69.89	64.71	66.95
ResNet50	53.12	66.30	62.76	60.36	64.90
DenseNet201	55.01	62.59	61.03	61.37	57.10
InceptionResNetV2	43.79	56.24	55.60	54.68	58.26
Xception	36.16	52.27	53.69	50.03	55.27

Table 5. Classification performance of the CNNs without data augmentation under transfer learning (5 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
Raw	56.29	74.40	79.53	72.92	78.46
VGG16	75.52	79.82	88.05	83.36	85.84
VGG19	76.23	80.67	88.86	83.67	86.47
ResNet50	74.91	84.35	78.73	82.39	80.43
DenseNet201	70.87	79.73	77.08	81.73	73.60
InceptionResNetV2	57.96	68.76	72.99	74.71	74.08
Xception	52.08	66.72	66.19	67.19	69.41

Table 6. Classification performance of the CNNs without data augmentation under transfer learning (4 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
Raw	46.40	69.94	74.51	70.31	74.99
VGG16	69.71	77.73	85.25	82.17	83.32
VGG19	70.71	78.64	86.34	82.63	84.51
ResNet50	78.25	88.50	86.99	83.46	87.08
DenseNet201	75.36	82.73	85.59	82.44	83.35
InceptionResNetV2	64.67	77.96	79.46	75.49	83.11
Xception	57.41	72.38	76.00	67.58	75.63

Next, the experiment was performed in the same way, but data augmentation was added for the training. The data were augmented with a rotation range of 10, a shear range of 10, a zoom range of 0.2, and a horizontal flip of “true”. The highest accuracies with data augmentation under transfer learning were 76.54% for VGG19 and type 2 preprocessing with the posture data of 11 classes, 67.47% for VGG19 and type 2 preprocessing with the posture data of 10 classes, 87.92% for VGG19 and type 2 preprocessing with the posture data of five classes, and 85.44% for DenseNet201 and type 2 preprocessing with the posture data of four classes. Table 7 presents the classification performance of the CNNs with data augmentation under transfer learning. Figure 18 shows the confusion matrix for VGG19 and type 2 preprocessing, which exhibited the best performance for the posture data of 11 classes with data augmentation under transfer learning. Tables 8–10 present the accuracies of the CNNs for the posture data of 10, 5, and 4 classes, respectively, with data augmentation under transfer learning.

Table 7. Classification performance of the CNNs with data augmentation under transfer learning (11 classes).

Single Model (%)		Type 0	Type 1	Type 2	Type 3	Type 4
SimpleCNN	Training	69.89	66.35	65.54	62.11	67.34
	Validation	62.92	59.47	63.02	43.42	61.97
	Testing	42.70	52.63	64.35	50.00	60.15
VGG16	Training	82.07	82.77	86.78	81.40	82.16
	Validation	73.73	73.02	76.40	72.37	74.29
	Testing	61.62	67.35	76.05	68.84	72.01
VGG19	Training	82.69	83.98	85.57	82.80	83.27
	Validation	73.13	73.21	76.74	72.64	74.14
	Testing	62.86	67.39	76.54	69.71	73.67
ResNet50	Training	58.33	67.89	57.76	66.10	62.82
	Validation	56.74	64.85	55.84	63.07	61.36
	Testing	53.43	60.67	49.63	63.04	54.15
DenseNet201	Training	68.33	71.84	73.12	67.96	67.00
	Validation	66.35	17.27	70.03	18.34	63.61
	Testing	57.04	66.41	64.25	61.08	54.24
InceptionResNetV2	Training	50.22	59.04	58.68	58.99	60.40
	Validation	48.10	15.05	56.78	15.15	58.42
	Testing	44.58	52.55	52.91	56.69	57.27
Xception	Training	44.31	57.97	54.08	55.36	57.00
	Validation	42.85	15.09	09.51	14.30	55.22
	Testing	35.15	52.54	45.63	53.53	50.94

Output Class	1	474 6.8%	5 0.1%	1 0.0%	12 0.2%	4 0.1%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	2 0.0%	0 0.0%	94.8% 5.2%
	2	71 1.0%	310 4.5%	3 0.0%	43 0.6%	13 0.2%	34 0.5%	0 0.0%	0 0.0%	0 0.0%	19 0.3%	4 0.1%	62.4% 37.6%
	3	9 0.1%	1 0.0%	392 5.7%	38 0.5%	23 0.3%	16 0.2%	3 0.0%	2 0.0%	7 0.1%	5 0.1%	4 0.1%	78.4% 21.6%
	4	19 0.3%	7 0.1%	36 0.5%	384 5.5%	22 0.3%	23 0.3%	0 0.0%	0 0.0%	1 0.0%	4 0.1%	4 0.1%	76.8% 23.2%
	5	0 0.0%	1 0.0%	28 0.4%	4 0.1%	377 5.4%	47 0.7%	4 0.1%	0 0.0%	6 0.1%	26 0.4%	4 0.1%	75.9% 24.1%
	6	1 0.0%	17 0.2%	11 0.2%	9 0.1%	117 1.7%	282 4.1%	2 0.0%	2 0.0%	2 0.0%	49 0.7%	6 0.1%	56.6% 43.4%
	7	0 0.0%	0 0.0%	5 0.1%	0 0.0%	2 0.0%	0 0.0%	277 4.0%	106 1.5%	86 1.2%	12 0.2%	1 0.0%	56.6% 43.4%
	8	3 0.0%	0 0.0%	9 0.1%	0 0.0%	7 0.1%	2 0.0%	93 1.3%	265 3.8%	86 1.2%	20 0.3%	4 0.1%	54.2% 45.8%
	9	0 0.0%	0 0.0%	4 0.1%	0 0.0%	4 0.1%	0 0.0%	105 1.5%	101 1.5%	229 3.3%	37 0.5%	2 0.0%	47.5% 52.5%
	10	0 0.0%	9 0.1%	7 0.1%	1 0.0%	24 0.3%	50 0.7%	9 0.1%	20 0.3%	12 0.2%	344 5.0%	5 0.1%	71.5% 28.5%
	11	3 0.0%	4 0.1%	6 0.1%	2 0.0%	6 0.1%	1 0.0%	1 0.0%	3 0.0%	0 0.0%	2 0.0%	1973 28.5%	98.6% 1.4%
		81.7% 18.3%	87.6% 12.4%	78.1% 21.9%	77.9% 22.1%	62.9% 37.1%	61.8% 38.2%	56.1% 43.9%	53.0% 47.0%	53.4% 46.6%	66.2% 33.8%	98.3% 1.7%	76.5% 23.5%
		Target Class											

Figure 18. Confusion matrix for VGG19 with preprocessing of type 2.

Table 8. Classification performance of the CNNs with data augmentation under transfer learning (10 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
SimpleCNN	24.20	38.35	49.95	40.17	46.41
VGG16	46.99	61.19	66.78	61.65	63.88
VGG19	48.94	60.45	67.47	62.40	65.12
ResNet50	48.85	64.00	58.69	58.22	59.42
DenseNet201	54.66	63.75	64.24	59.67	60.11
InceptionResNetV2	46.22	55.96	53.21	53.93	59.68
Xception	38.02	54.36	50.43	47.67	53.82

Table 9. Classification performance of the CNNs with data augmentation under transfer learning (5 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
Raw	51.43	60.43	78.71	61.96	73.37
VGG16	72.37	78.44	87.52	82.19	83.76
VGG19	74.44	78.19	87.92	82.49	84.83
ResNet50	69.54	78.70	73.47	78.49	75.94
DenseNet201	71.00	80.70	81.17	79.79	75.83
InceptionResNetV2	61.67	71.71	73.74	72.34	77.13
Xception	52.50	68.65	66.38	66.55	68.93

Table 10. Classification performance of the CNNs with data augmentation under transfer learning (4 classes).

Single Model (%)	Type 0	Type 1	Type 2	Type 3	Type 4
Raw	42.31	53.72	73.55	58.97	68.33
VGG16	66.13	77.50	84.75	81.15	81.78
VGG19	68.83	76.71	85.24	81.22	82.46
ResNet50	70.85	85.37	85.05	79.46	84.69
DenseNet201	73.14	82.72	85.44	83.68	84.92
InceptionResNetV2	67.04	78.69	79.21	74.61	83.64
Xception	58.73	73.89	73.76	66.28	75.34

The CNNs were trained via nontransfer learning. VGG19, ResNet50, ResNet101, and InceptionResNetV2 were considered for nontransfer learning. The first three models were trained with a batch size of 60, three epochs, and the Adam optimizer [43]; the last model was trained with a batch size of 20, one epoch, and the Adam optimizer. The highest accuracies were 93.32% for InceptionResNetV2 and type 2 preprocessing with the posture data of 11 classes without data augmentation under nontransfer learning. Table 11 presents the classification performance for 11 classes without augmentation under nontransfer learning. Table 12 presents the training time of a single model for 11 classes without augmentation under nontransfer learning. Figure 19 shows the training process for InceptionResNetV2 and type 2 preprocessing, which exhibited the best performance for the posture data of 11 classes without data augmentation under nontransfer learning. Figure 20 shows the activations for InceptionResNetV2 and type 2 preprocessing. Figure 21 shows the activations of the last ReLU for ResNet50 and type 2 preprocessing.

Table 11. Classification performance for 11 classes without augmentation under nontransfer learning.

Single Model (%)		Type 0	Type 1	Type 2	Type 3	Type 4
VGG19	Training	91.67	90.99	91.20	89.88	87.37
	Validation	90.42	90.19	89.76	88.79	86.75
	Testing	88.40	87.76	90.22	88.29	87.06
ResNet50	Training	95.47	96.37	96.16	95.60	95.62
	Validation	92.70	92.30	93.54	92.04	92.54
	Testing	89.99	88.77	91.78	88.43	90.48
ResNet101	Training	96.81	96.02	96.11	95.16	95.58
	Validation	93.86	92.44	92.90	91.97	91.94
	Testing	90.93	88.04	91.29	88.32	89.17
InceptionResNetV2	Training	95.15	95.25	95.65	92.80	94.31
	Validation	93.78	91.89	94.01	92.04	91.90
	Testing	92.54	88.98	93.32	89.50	91.09

Table 12. Training time of a single model for 11 classes without augmentation under nontransfer learning.

Single Model (hh:mm)		Type 0	Type 1	Type 2	Type 3	Type 4
VGG19	Training	26:35	28:38	27:04	27:37	28:10
ResNet50	Training	10:39	12:22	12:09	08:02	08:45
ResNet101	Training	25:46	27:05	27:30	26:09	27:01
InceptionResNetV2	Training	53:23	57:54	59:09	58:25	61:44

Figure 19. Training process for InceptionResNetV2 with preprocessing of type 2: (a) accuracy; (b) loss.

Figure 20. Activations for InceptionResNetV2 with preprocessing of type 2, which exhibited the best performance for the posture data of 11 classes without data augmentation under nontransfer learning: (a) activations of first convolution; (b) activations of last convolution.

Figure 21. Activations of the last ReLU for InceptionResNetV2 with preprocessing of type 2, which exhibited the best performance for the posture data of 11 classes without data augmentation under nontransfer learning.

The trained CNN models from Table 11 are combined using the majority of outputs and average of scores. First, the five CNNs trained with different input types are combined under VGG19, ResNet50, ResNet101, and InceptionResNetV2 as EV19TNet, ER50TNet, ER101TNet, and EIR2TNet, respectively. Table 13 describes the classification performance of ensemble deep models designed by input types. Secondly, the four CNNs trained with different models are combined under input types from 0 to 4 as ET0MNet, ET1MNet, ET2MNet, ET3MNet, and ET4MNet, respectively. Table 14 indicates the classification performance of ensemble deep models designed by the pre-trained CNNs. Table 15 indicates the training time of the ensemble system for 11 classes without augmentation under nontransfer learning. In ensemble systems, the highest accuracy is 95.34% of InceptionResNetV2s using average of scores.

Table 13. Classification performance of ensemble deep models designed by input types.

Ensemble System		Majority	Average
EV19TNet	Testing	92.77	93.42
ER50TNet	Testing	93.70	94.12
ER101TNet	Testing	93.94	94.51
EIR2TNet	Testing	94.76	95.34

Table 14. Classification performance of ensemble deep models designed by the pre-trained CNNs.

Ensemble System		Majority	Average
ET0MNet	Testing	93.08	94.07
ET1MNet	Testing	91.53	92.77
ET2MNet	Testing	93.94	94.58
ET3MNet	Testing	91.30	92.31
ET4MNet	Testing	92.86	93.67

Table 15. Training time of ensemble deep models for 11 classes without augmentation under nontransfer learning.

Ensemble System (hh:mm)		Majority
EV19TNet	Training	138:04
ER50TNet	Training	51:57
ER101TNet	Training	133:31
EIR2TNet	Training	290:35
ET0MNet	Training	116:23
ET1MNet	Training	125:59
ET2MNet	Training	125:52
ET3MNet	Training	120:13
ET4MNet	Training	125:40

The classification performances listed in Tables 3–10 were computed by average value. Figure 22 shows the average values of classification performance under transfer learning for the different types of preprocessing. The experimental results on preprocessing of type 2 showed 13.63%, 3.12%, 2.79%, and 0.76% higher classification rate than types 0, 1, 3, and 4, respectively. Figure 23 shows a comparison of the average total accuracies for the different models under transfer learning. The ‘SimCNN’, ‘DenseNet’, and ‘IncResNet’ are simple CNN, DenseNet201, and InceptionResNetV2, respectively. The experimental results by VGG19 showed 15.78%, 0.78%, 3.53%, 4.11%, 10.61%, and 16.02% higher classification performance than the simple CNN, VGG16 [19], ResNet50 [20], DenseNet201 [21], InceptionResNetV2 [22], and Xception [23], respectively.

Figure 22. Comparison of the average total accuracies for the different types of preprocessing under transfer learning.

Figure 23. Comparison of the average total accuracies for the different models under transfer learning.

Figure 24 visualizes the classification performance of ensemble deep models from the results listed in Tables 13 and 14. Here, 'BestSingle' is the best result of single models listed in Table 11. As shown in Figure 24, the ensemble deep model by InceptionResNetV2s with average method showed the best classification performance in comparison to other models.

Figure 24. Performance comparison by various ensemble deep models.

5. Conclusions

We present ensemble deep models designed by pre-trained CNNs and various types of preprocessing for posture recognition under various home environments. The pertained CNNs used in this paper were performed by VGGNet, ResNet, DenseNet, InceptionResNet, and Xception, frequently used in conjunction with deep learning. Finally, we performed systematic experiments from a large posture database constructed by the Electronics and Telecommunications Research Institute (ETRI). The experimental results reveal that the ensemble deep model shows good performance in comparison with the pre-trained CNN itself. Thus, we expect that the presented method will be an important technique to help elderly people from sudden danger in home environments. In future research, we shall perform the research on the behavior recognition of elderly people from a large 3D database for behavior recognition constructed under home service robot environments.

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

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2017R1A6A1A03015496). This work was supported by the ICT R&D program of MSIT/IITP (2017-0-00162, Development of Human-care Robot Technology for Aging Society).

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

References

1. Park, J.H.; Song, K.H.; Kim, Y.S. A kidnapping detection using human pose estimation in intelligent video surveillance systems. *J. Korea Soc. Comput. Inf.* **2018**, *23*, 9–16.
2. Qiang, B.; Zhang, S.; Zhan, Y.; Xie, W.; Zhao, T. Improved convolutional pose machines for human pose estimation using image sensor data. *Sensors* **2019**, *19*, 718. [[CrossRef](#)] [[PubMed](#)]
3. Huang, Z.; Liu, Y.; Fang, Y.; Horn, B.K.P. Video-Based Fall Detection for Seniors with Human Pose Estimation. In Proceedings of the 4th International Conference on Universal Village, Boston, MA, USA, 21–24 October 2018; pp. 1–4.
4. Chan, K.C.; Koh, C.K.; Lee, C.S.G. An automatic design of factors in a human-pose estimation system using neural networks. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *46*, 875–887. [[CrossRef](#)]
5. Veges, M.; Varga, V.; Lorincz, A. 3D human pose estimation with siamese equivariant embedding. *Neurocomputing* **2019**, *339*, 194–201. [[CrossRef](#)]
6. Stommel, M.; Beetz, M.; Wu, W. Model-free detection, encoding, retrieval, and visualization of human poses from Kinect data. *IEEE Asme Trans. Mechatron.* **2015**, *20*, 865–875. [[CrossRef](#)]

7. Shum, H.P.H.; Ho, E.S.L.; Jiang, Y.; Takagi, S. Real-time posture reconstruction for Microsoft Kinect. *IEEE Trans. Cybern.* **2013**, *43*, 1357–1369. [CrossRef]
8. Lee, J.; Joo, H.; Lee, J.; Chee, Y. Automatic classification of squat posture using inertial sensors: Deep learning approach. *Sensors* **2020**, *20*, 361. [CrossRef]
9. Chowdhury, I.R.; Saha, J.; Chowdhury, C. Detailed Activity Recognition with Smartphones. In Proceedings of the Fifth International Conference on Emerging Applications of Information Technology, Kolkata, India, 12–13 January 2018; pp. 1–4.
10. Wu, Z.; Zhang, J.; Chen, K.; Fu, C. Yoga posture recognition and quantitative evaluation with wearable sensors based on two-stage classifier and prior bayesian network. *Sensors* **2019**, *19*, 5129. [CrossRef]
11. Idris, M.I.; Zabidi, A.; Yassun, I.M.; Ali, M.S.A.M. Human Posture Recognition Using Android Smartphone and Artificial Neural Network. In Proceedings of the IEEE Control and System Graduate Research Colloquium, Shah Alam, Malaysia, 10–11 August 2015; pp. 120–124.
12. Pak, M.S.; Kim, S.H. A Review of Deep Learning in Image Recognition. In Proceedings of the International Conference on Computer Applications and Information Processing Technology, Kuta Bali, Indonesia, 8–10 August 2017; pp. 1–3.
13. Diao, W.; Sun, X.; Zheng, X.; Dou, F.; Wang, H.; Fu, K. Efficient saliency-based object detection in remote sensing images using deep belief networks. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 137–141. [CrossRef]
14. Sasaki, H.; Horiuchi, T.; Kato, S. A Study on Vision-Based Mobile Robot Learning by Deep Q-Network. In Proceedings of the Annual Conference of Society of Instrument Control Engineers, Kanazawa, Japan, 19–22 September 2017; pp. 799–804.
15. Chang, C.H. Deep and shallow architecture of multilayer neural networks. *IEEE Neural Netw. Learn. Syst.* **2015**, *26*, 2477–2486. [CrossRef]
16. Mou, L.; Ghamisi, P.; Zhu, X.X. Deep recurrent neural networks for hyperspectral image classification. *IEEE Geosci. Remote Sens.* **2017**, *55*, 3639–3655. [CrossRef]
17. Callet, P.L.; Viard-Gaudin, C.; Barba, D. A convolutional neural network approach for objective video quality assessment. *IEEE Neural Netw.* **2006**, *17*, 1316–1327. [CrossRef] [PubMed]
18. Hou, J.C.; Wang, S.S.; Lai, Y.H.; Tsao, Y.; Chang, H.W.; Wang, H.M. Audio-visual speech enhancement using multimodal deep convolutional neural networks. *IEEE Emerg. Top. Comput. Intell.* **2018**, *2*, 117–128. [CrossRef]
19. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2015**, arXiv:1409.1556v6. Available online: <https://arxiv.org/abs/1409.1556> (accessed on 5 December 2019).
20. 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, 27–30 June 2016; pp. 770–778.
21. Huang, G.; Liu, Z.; Maaten, L.V.D.; Weinberger, K.Q. Densely connected convolutional networks. *arXiv* **2018**, arXiv:1608.06993v5. Available online: <https://arxiv.org/abs/1608.06993> (accessed on 5 December 2019).
22. Szegedy, C.; Ioffe, S.; Vanhoucke, V. Inception-v4, Inception-ResNet and the impact of residual connections on learning. *arXiv* **2016**, arXiv:1602.07261v2. Available online: <https://arxiv.org/abs/1602.07261> (accessed on 5 December 2019).
23. Chollet, F. Xception: Deep learning with depthwise separable convolutions. *arXiv* **2017**, arXiv:1610.02357v3. Available online: <https://arxiv.org/abs/1610.02357> (accessed on 8 December 2019).
24. Shao, L.; Zhu, F.; Li, X. Transfer learning for visual categorization: A survey. *IEEE Neural Netw. Learn. Syst.* **2015**, *26*, 1019–1034. [CrossRef]
25. Tompson, J.; Jain, A.; LeCun, Y.; Bregler, C. Joint training of a convolutional network and a graphical model for human pose estimation. *arXiv* **2014**, arXiv:1406.2984v2. Available online: <https://arxiv.org/abs/1406.2984> (accessed on 8 December 2019).
26. Carreira, J.; Agrawal, P.; Fragkiadaki, K.; Malik, J. Human pose estimation with iterative error feedback. *arXiv* **2016**, arXiv:1507.06550v3. Available online: <https://arxiv.org/abs/1507.06550> (accessed on 8 December 2019).
27. Pishchulin, L.; Insafutdinov, E.; Tang, S.; Andres, B.; Andriluka, M.; Gehler, P.; Schiele, B. DeepCut: Joint subset partition and labeling for multi person pose estimation. *arXiv* **2016**, arXiv:1511.06645v2. Available online: <https://arxiv.org/abs/1511.06645> (accessed on 8 December 2019).
28. Girshick, R. Fast R-CNN. *arXiv* **2015**, arXiv:1504.08083v2. Available online: <https://arxiv.org/abs/1504.08083> (accessed on 8 December 2019).

29. Insafutdinov, E.; Pishchulin, L.; Andres, B.; Andriluka, M.; Schiele, B. DeeperCut: A deeper, stronger, and faster multi-person pose estimation model. *Adv. Concepts Intell. Vis. Syst.* **2016**, *9910*, 34–50.
30. Georgakopoulos, S.V.; Kottari, K.; Delibasis, K.; Plagianakos, V.P.; Maglogiannis, I. Pose recognition using convolutional neural networks on omni-directional images. *Neurocomputing* **2018**, *280*, 23–31. [[CrossRef](#)]
31. Liu, Y.; Xu, Y.; Li, S.B. 2-D Human Pose Estimation from Images Based on Deep Learning: A Review. In Proceedings of the 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, Xi'an, China, 25–27 May 2018; pp. 462–465.
32. Lee, J.N.; Byeon, Y.H.; Kwak, K.C. Design of ensemble stacked auto-encoder for classification of horse gaits with MEMS inertial sensor technology. *Micromachines* **2018**, *9*, 411. [[CrossRef](#)] [[PubMed](#)]
33. Maguolo, G.; Nanni, L.; Ghidoni, S. Ensemble of convolutional neural networks trained with different activation features. *arXiv* **2019**, arXiv:1905.02473v4. Available online: <https://arxiv.org/abs/1905.02473> (accessed on 10 December 2019).
34. Kim, M.G.; Pan, S.B. Deep learning based on 1-D ensemble networks using ECG for real-time user recognition. *IEEE Trans. Ind. Inform.* **2019**, *15*, 5656–5663. [[CrossRef](#)]
35. Kahlouche, S.; Ouadah, N.; Belhocine, M.; Boukandoura, M. Human Pose Recognition and Tacking Using RGB-D Camera. In Proceedings of the 8th International Conference on Modelling, Identification and Control, Algiers, Algeria, 15–17 November 2016; pp. 520–525.
36. Na, Y.J.; Wang, C.W.; Jung, H.Y.; Ho, J.G.; Choi, Y.K.; Min, S.D. Real-Time Sleep Posture Recognition Algorithm Using Kinect System. In Proceedings of the Korean Institute of Electrical Engineers Conference on Biomedical System, Hoengseong, Korea, 15–17 February 2016; pp. 27–30.
37. Kim, S.C.; Cha, J.H. Posture Recognition and Spatial Cognition with Hybrid Sensor. In Proceedings of the Korean Society of Precision Engineering Conference, Jeju, Korea, 29–31 May 2013; pp. 971–972.
38. Giancola, S.; Valenti, M.; Sala, R. A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies. In *Springer Briefs in Computer Science*; Zdonik, S., Shekhar, S., Wu, X., Jain, L.C., Padua, D., Shen, X.S., Furht, B., Subrahmanian, V.S., Hebert, M., Ikeuchi, K., et al., Eds.; Springer Nature: Cham, Switzerland, 2018; pp. 29–39. ISBN 9783319917603.
39. Redmon, J.; Farhadi, A. YOLOv3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767v1. Available online: <https://arxiv.org/abs/1804.02767> (accessed on 10 December 2019).
40. Choi, H.S.; Lee, B.H.; Yoon, S.R. Biometric authentication using noisy electrocardiograms acquired by mobile sensors. *IEEE Access* **2016**, *4*, 1266–1273. [[CrossRef](#)]
41. RMSprop Optimization Algorithm for Gradient Descent with Neural Networks. Available online: <https://insidebigdata.com/2017/09/24/rmsprop-optimization-algorithm-gradient-descent-neural-networks/> (accessed on 16 May 2019).
42. Zeiler, M.D. Adadelta: An adaptive learning rate method. *arXiv* **2012**, arXiv:1212.5701v1. Available online: <https://arxiv.org/abs/1212.5701> (accessed on 13 December 2019).
43. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2017**, arXiv:1412.6980v9. Available online: <https://arxiv.org/abs/1412.6980> (accessed on 13 December 2019).