applied sciences

MDPI

Article

# Compact Spatial Pyramid Pooling Deep Convolutional Neural Network Based Hand Gestures Decoder

**Akm Ashiquzzaman [1], Hyunmin Lee [2,*], Kwangki Kim [3,*], Hye-Young Kim [4], Jaehyung Park [1] and Jinsul Kim [1,*]**

1 Department of ICT Convergence System Engineering, Chonnam National University, Gwangju 61186, Korea; zamanashiq3@chonnam.ac.kr (A.A.); hyeoung@jnu.ac.kr (J.P.)
2 Human IT Convergence Research Center, Korea Electronics Technology Institute, Gyeonggi-do 13509, Korea
3 School of IT Convergence, Korea Nazarene University, Chungcheongnam-do 31172, Korea
4 School of Game/Game Software, Hongik University, Seoul 04066, Korea; hykim@hongik.ac.kr
* Correspondence: hyunmw@keti.re.kr (H.L.); k2kim@kornu.ac.kr (K.K.); jsworld@jnu.ac.kr (J.K.)

Received: 14 September 2020; Accepted: 3 November 2020; Published: 7 November 2020


**Abstract:** Current deep learning convolutional neural network (DCNN) -based hand gesture detectors with acute precision demand incredibly high-performance computing power. Although DCNN-based detectors are capable of accurate classification, the sheer computing power needed for this form of classification makes it very difficult to run with lower computational power in remote environments. Moreover, classical DCNN architectures have a fixed number of input dimensions, which forces preprocessing, thus making it impractical for real-world applications. In this research, a practical DCNN with an optimized architecture is proposed with DCNN filter/node pruning, and spatial pyramid pooling (SPP) is introduced in order to make the model input dimension-invariant. This compact SPP-DCNN module uses 65% fewer parameters than traditional classifiers and operates almost 3× faster than classical models. Moreover, the new improved proposed algorithm, which decodes gestures or sign language finger-spelling from videos, gave a benchmark highest accuracy with the fastest processing speed. This proposed method paves the way for various practical and applied hand gesture input-based human-computer interaction (HCI) applications.

**Keywords:** deep learning; convolutional neural network; hand gesture recognition; neural network pruning; optimization

## 1. Introduction

Hand gesture recognition is a key part of human communication. Such gestures were the primary means of communication in the prehistoric age [1]. In modern days, hand gestures are still useful, for example, in the case of difficulties with oral communication or human-computer interactions (HCIs). Today, visual experience plays a significant part in HCIs. An interactive and properly gesture-classifiable computer program can properly recognize human behavior as an input for processing. The use of sign language as a gesture-based, as opposed to voice-based, means of communication with another medium makes HCI tremendously promising. The most natural form of gesture-based communication is sign language. Hearing-impaired individuals and people with speech impediments use sign language in their everyday lives. American Sign Language (ASL), which is the most popular form of sign language in the world, is derived from the old French hand symbols and used throughout the continent of North America. Sign language, first developed two centuries ago, is now widely recognized throughout the world. Visual perception plays a vital role in HCI [2]. HCI bridges the gap between machines and humans, allowing them to communicate with each other. Hence, naturally, hand gesture

recognition has been the subject of a significant amount of research in both the communication and machine learning domains. The main idea in the machine learning domain for HCI is to properly classify and detect human gestures and improve the accuracy of the classifier.

The main drawback to implementing any kind of machine learning or computer vision-based gesture classification involves the detection and classification of different hand gestures. Theoretically, there are a limited number of hand gestures, and mapping those to the almost infinite number of possible expressions is close to impossible. For this reason, most computer vision-based research solely focuses on classification of a limited number of gestures. On the other hand, HCI studies concern the development of deep learning-based classification and hand gesture detection systems. Deep convolutional neural networks (DCNNs) have recently gained popularity because of their superior ability to recognize visual data. Utilizing DCNNs for object classification is not a recent advancement. Much research has been done, solely focusing on the DCNN-based classification of hand gestures.

DCNNs are widely used machine learning algorithms that automate feature extraction and classification without the need for external selective feature extraction methods. This is done by the two (2) main steps executed by a DCNN algorithm. The convolutional layers in a DCNN utilize convolutional operations in order to extract the important features of input images. Subsequently the fully connected or dense layer of the DCNN classifies the features and produces a prediction of the given input images. The use of a DCNN eliminates the need for human or machine-based feature selection criteria and improves the overall accuracy of classification performance.

However, DCNNs have certain fundamental limitations for hand gesture classification. By definition, the input image that is presented to the convolutional layers of a DCNN must have a fixed number of dimensions. This creates problems that are related to resizing or cropping of the input images and results in reduced accuracy and mislabelling due to data loss. Moreover, most research on DCNNs has solely focused on achieving accuracy. Thus, DCNN layers are increasing in size, number, and complexity over time. This means that practical applications of DCNNs are extremely power and computational resource-intensive (in terms of CPU, GPU, RAM, etc.).

Addressing high computational costs and optimizing neural networks is a well-researched topic. Many DCNN optimization techniques have been presented in recent years. DCNN node pruning is one such optimization technique, first suggested by Lecun et al. [3]. Neural node and convolution filter pruning is the process of eliminating neural network nodes and filters that are based on rational or mathematical comparisons. Although this process is not new, applying it to modern DCNNs is a comparatively new development. Spatial pyramid pooling (SPP) is a DCNN convolutional layer feature gathering method that eliminates the need for fixed-dimensional input. Hu et al. first described this process for a convolutional neural network [4]. To the best of our knowledge, the combination of both of these techniques has not yet been studied.

This research addresses the ideas of DCNN node pruning and introduction of an SPP layer in the DCNN, and uses several metrics to evaluate the performance of the subsequent model. Also, a new algorithm that decodes hand gestures in real time for practical uses is proposed and analyzed. The rest of the article is organized as follows. Section 2 describes previous work done in this field that inspired this research. Section 3 discusses the fundamentals of DCNN node pruning and SPP. Our proposed method is presented in Section 4. Section 5 describes the dataset that was used to train and test the proposed method. Section 6 concerns the evaluation metrics that were used to validate and test the results presented in Section 7. Finally, we compare the proposed system with other notable work in Section 8 and, finally, provide a summary of this work and future directions in Section 9.

## 2. Related Works

In recent decades, many researchers have investigated different approaches that use touch and non-touch gestures in various real-world practical applications. The practicality of manufacturing such systems has been the subject of many experiments in the last few years. Because of the nature of hand gestures and the specifics of machine learning research, discussing all of this a gargantuan task. Accordingly, this section will briefly discuss the original research on machine learning hand gesture classification, followed by the application of DCNNs and the potential drawbacks of such methods.

Earlier research into both machine learning and computer vision-based ASL recognition relied heavily on traditional feature selection methods. These methods involved processing input images into various transformations and then selecting the best features for to classifying the ASL symbol represented in the image. Vogler et al. presented notable work that utilized the above-mentioned methods. Their work used an adaptive hidden Markov model (HMM) and three-dimensional (motion) analysis for recognition [5,6]. These types of studies are usually expensive, as they require additional motion capture instrumentation that is both time- and resource-intensive. The development of a more practical setup to overcome such limitations must follow a vision learning-based approach.These types of studies are usually expensive, as they require additional motion capture instrumentation that is both time- and resource-intensive. The development of a more practical setup to overcome such limitations must follow a vision learning-based approach.

A variety of approaches to computer vision-based ASL detection have been developed [7]. These include both traditional machine learning- and deep learning-based approaches. Notable work in traditional machine learning approaches includes singular value decomposition (SVD) [8], wavelet transformation via discrete wavelet transformation (DFT) [9], geometric features, and local binary patterns (LBP) [10]. These methods rely heavily on extraction of input images while using varieties of classifiers, including but not limited to ensemble classifiers that are based on support vector machines (SVMs) [11], artificial neural networks (ANNs) [12], and linear regression and genetic algorithms (GAs) [13]. These methods produced a significant improvement in both image- and sensor-based input processing. Hand gesture image feature extraction has many applications, some of which rely on Gabor filters [14]. This research follows a common theme of input data preprocessing and feature extraction systems. Although much of this research reported high accuracy, the preprocessing and real-time processing requirements for input data extraction make these systems very impractical to deploy as real-time gesture recognition applications. In the middle of the 20th century, Hubel and Weisel explained the biological underpinnings of visual sensory processing in the mammalian context [15]. In essence, they showed that a small amount of edge detection can lead to superior image detection. This research served as a precursor to DCNN-based research in computer vision. Lecunn et al. [16] were the first to successfully demonstrate neural convolutional network training with backpropagation. Krizhevsky et al. [17], in work that won the 2012 Imagenet competition, opened the door to use of DCNNs in different image detection applications. Their CNN introduced deep feed-forward artificial neural network, which is now the most popular visual image analysis technique. It has been used with exceptional precision in a number of contexts in different models of object recognition and classification [18].

A great deal of previous work has been focused on DCNN-based sign recognition [19,20]. Some studies used a sensor-only approach [21] . An optimized form of Microsoft's Kinect device [22,23], which is primarily a gesture feedback interface for gaming, can be used for gesture classification input data collection [24]. The classic DCNN approach to image-based hand gesture classification has also been studied, with a sole focus on accuracy [25]. This has resulted in some extremely complex models with a very high number of parameters which often require data resizing or background subtraction to achieve an acceptable degree of accuracy [26,27]. Waeerasekera et al. [10] classified this problem as the ASL based finger-spelling recognition/classification problem. However, the underlining machine learning mechanism of ASL based finger-spelling and the gesture recognition is indifferent due to the invariant dataset.Both high network size and input dimension restrictions limit the applicability

of DCNN algorithms. Hand gesture recognition must be robust and input dimension-invariant, as the same hand gesture can appear in different scenarios, which results in a variation in the number of input dimensions. Moreover, DCNN algorithms need to be faster and computationally less expensive if they are to be deployed in practical scenarios. We propose a new algorithm that utilizes *L*2-normalization-based DCNN node and filter pruning and introduce a spatial pyramid pooling layer (SPP) that eliminates the necessity to restrict the number of input dimensions in order to address the data dimension restrictions and computational complexity of DCNNs. We also propose a new real-time detection hand gesture classifier that can recognize and decode multiple gestures in real time from an input video. We also utilized the open sourced ASL dataset [28], which provided 29 separate hand gesture classes for robust ASL finger-spelling or gesture recognition.

## 3. Basic Theory for Optimizing DCNN

### *3.1. Deep Convolutional Neural Network and Node Pruning*

The basic concept of a DCNN originated from the theory that an input matrix that controls edge features can be used to understand and define the patterns that appear. Any affine transformation cannot be used to obtain valuable visual information. Many previous work and literates have described the basic DCNN theory thoroughly [29,30]. Convolution is a sparse operation, and the parameters are reused by sharing. This will extract the edge or essential pattern information from visual feedback and label it as the supplied data. Figure 1 shows a simplified DCNN with 2 (two) convolutional layers, followed by a flattenws layer with the extracted feature vector that are used to classify the given image input. Convolution is a mathematical operation that can be applied to a matrix. Basically, all of the images that serve as the input for a convolutional neural network can be represented as an *N*-dimensional matrix. The filter sub-matrix is the underlying weight of the neural network and the inputs are integrated across the channels, which are the image dimensions.

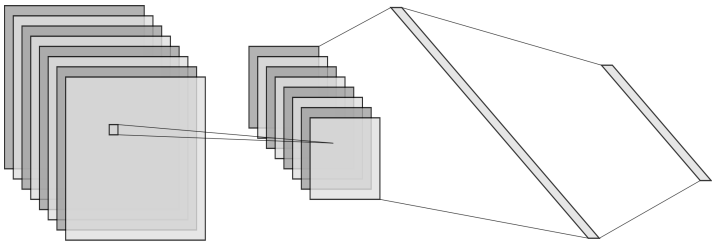

**Figure 1.** Simplified deep convolutional network with two (2) convolutional layers and one (1) single fully connected layer.

If a single node of a neural network can be denoted as the $y$, which is the mathematical summation of $i$ inputs $x$ and weights, the node can be expressed as the following matrix form,

$$y = \sum_{i=1}^{n} w_i x_i + b \tag{1}$$

Here, $b$ is the bias, and to reduce the result to a threshold based non-linearity, an activation function $\sigma(x)$ can be used, as follows,

$$\hat{y} = \sigma(y) \tag{2}$$

Accordingly, the convolution accepts a volume size of $W$, $H$, $D$ as the weight, height, and dept the number of filters $K$ their stride $S$, and followed by the spatial extent $F$ and calculates the following,

$$\begin{aligned} \hat{W} &= \frac{W - F + 2}{S + 1} \\ \hat{H} &= \frac{H - F + 2}{S + 1} \\ \hat{D} &= K \end{aligned} \tag{3}$$

Here, $\hat{W}, \hat{H}, \hat{D}$ are the resulting output matrix dimensions. It is important to mention that the weight and height are computed equally by symmetry and the output convoluted matrix obtains the exact filter-sized volume for the next convolutional layer to process. Thus, based on the equation that is shown above, a simple ranking on the nodes based on the ascending values of $\hat{W}$ can be used in order to rank the filters of each layer in both the convolutional and fully connected layers. $l1$ normalization of the weight vectors can be calculated as the element-wise vector absolute value, as shown below,

$$l1(\hat{W}) \ = \ |\hat{W}| \tag{4}$$

Subsequently, the nodes can be sorted from the calculated $l1$ normalization and pruned based on the least scored nodes, as needed. Figure 2 displays the block representation of node pruning in one particular layer of the DCNNs.

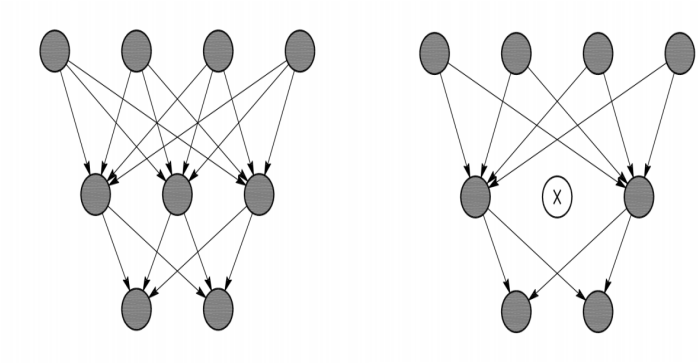

**Figure 2.** Basic concept of node/filter pruning in a deep convolutional neural network.

### *3.2. Image Resizing Restriction in DCNN*

A DCNN, by definition, must be given a fixed number of input image dimensions, which depends on the number with which it was trained. For example, Lenet∗5, which was trained on the MNIST dataset, requires input dimensions of $(32, 32)$ [31]. This means that all of the DCNN developments based on the MNIST dataset are forced to resize any model to the aforementioned dimensions. Figure 3 demonstrates the problem that arises when resizing input images.

Moreover, the Imagenet challenge dataset has a fixed $(224, 224)$ input dimension restriction [32]. Consequently, most DCNN architectures developed in recent years require those same dimensions. A trained DCNN model might incorrectly classify images that have been resized from a different number of dimensions. In Figure 3, the original $(224, 224)$ input images has been resized into 2 different images of $(200, 400)$ and $(400, 200)$. This results in DCNN wrongly classifying the same images with different classes, making this DCNN unreliable to use in practical life, where the input dimension varies heavily. This is the main motivation to incorporate the spatial pyramid pooling (SPP) layer into the DCNN.

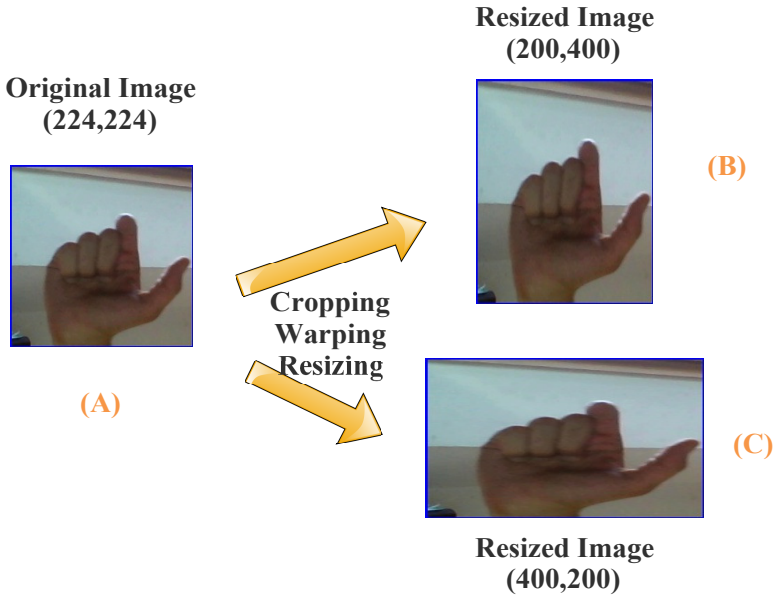


**Figure 3.** Image resizing cause distortion of spatial information, resulting mis-classification and restricting deep learning convolutional neural network (DCNN) input dimensions. Here, (**A**) is the original image, resizing it to feed into DCNN cause distortion shown in (**B**,**C**).

### *3.3. Spatial Pyramid Pooling Layer (SPP)*

The use of a spatial pyramid pooling (SPP) layer eliminates the input image dimension constraints of DCNNs. SPPs are based on original work more commonly referred to as spatial pyramid matching (SPM). SPM is an extension of the bag-of-words (BoW) model that was proposed by Sivic et al. [33], a classical computer vision algorithm that divides input image feature vectors into finer-to-coarser forms or sections. Later, the algorithm aggregates feature maps in the sections. SPP not only helps to produce representations for processing from uniformly scaled images/windows, but also helps to feed the DCNN images of different sizes or scales during image preparation. Training with images of variable size improves the scale-invariance and eliminates the over-fitting of a DCNN. He et al. demonstrated that integration of SPP into a DCNN design improves accuracy in traditional DCNN architectures, such as Lenet∗5, Alexnet with the Imagenet dataset, etc.

Figure 4 shows the main workings of the SPP layer in the DCNN. In the block diagram, (A) is the input image with any arbitrary input. The image is put through the convolutional feature pooling layer in (B). The features pulled by the convolutional layer are passed to the SPP layer. SPP generates a fixed-length output regardless of the size of the input, whereas the common DCNN sliding window pooling used in normal deep networks cannot perform this operation (C). The operation improves upon BoW models, such that the network retains spatial information by performing max-pooling in local spatial bins. The sizes of these spatial bins are proportional to the image size, so the number of bins is fixed regardless of the image size. This is in contrast to the sliding window pooling used in classical DCNN approaches, in which each spatial bin reflects the max-pooling responses of each filter. The outputs of SPP are presented as $kM$-dimensional vectors, where $M$ denotes the number of bins and $k$ is the number of filters in the last convolutional layer. Figure 4D shows this process. These vectors are then transferred to the fully connected layers for classification (Figure 4E) and display (Figure 4F).

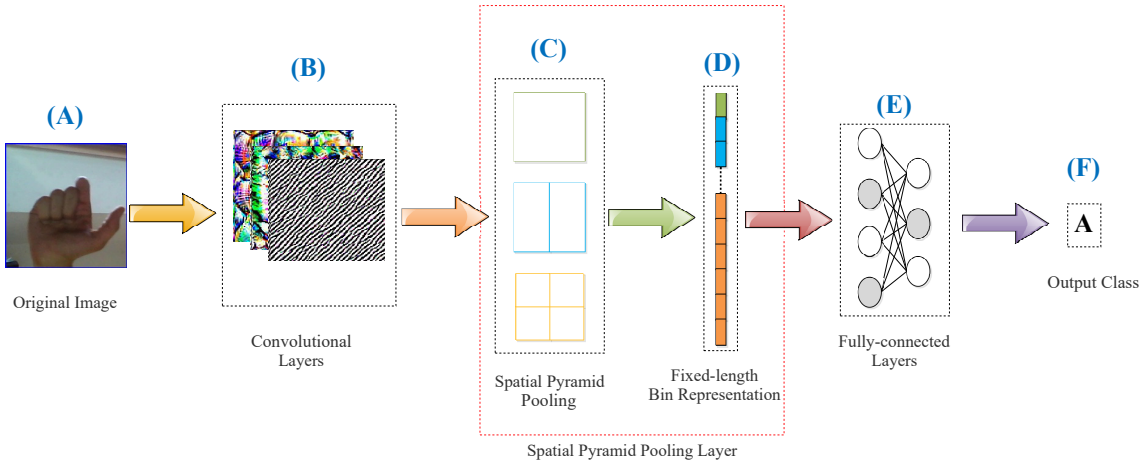


**Figure 4.** Spatial pyramid pooling in a DCNN. (**A**) A block diagram input is subjected to (**B**) convolutional feature pooling, then (**C**) max-pooled spatial pyramid pooling (SPP) with (**D**) a fixed number of bins; the result is finally transferred to (**E**) the fully connected (FC) layer, which (**F**) classifies the results.

## 4. The Proposed Methodology

This section of the article discusses the implementation of the details of the proposed method. For the sake of simplicity, we separate the proposed method into two main sections. The first section deals with selection of the optimal nodes in the DCNN by pruning. The second section discusses transformation of the DCNN to a compact SPP-based DCNN, along with a practical approach to decoding video gestures in real time based on the compact SPP-based DCNN.

### *4.1. Practical DCNN Architecture Selection and Pruning Strategy for Optimal Node Selection*

Although SPP is not a modern concept in relation to DCNN, the use of both node pruning and transformation from a classical DCNN to an SPP-based DCNN has not yet been thoroughly researched. Additionally, we accomplish single image-based decoding by using traditional methods in a new manner; our technique makes the most of other approaches that have until now not been practical to deploy in real-life settings. Figure 5 demonstrates the simplicity of the proposed method. Almost all general feature selection-based classifiers carry out resizing and transformation from red, green, blue (RGB)-scale to gray-scale for classification, whereas the DCNN with SPP only requires a dimension-invariant input image for classification.

The proposed DCNN architecture was constructed based on research by Oxford's Vision Geometry Group (VGG) [34]. Their proposed model DCNN has several versions for deployment. We started with the smallest DCNN in VGGnet and pruned from that architecture, as our original dataset contains far fewer labels than the Imagenet dataset used to train VGGnet. The main goal of this pruning is optimization and creation of a compact model that is less computationally intensive. The smallest original VGGnet model is the VGGnet-11. The numeral eleven $(11)$ represents the total number of hidden convolutional and fully connected layers. Figure 6 shows the full network structure and node/filter for each layer. The input layer of the DCNN takes a fixed $(224, 224)$ RGB input, and it is followed by the first convolutional layer, with 64 units/filters. Then, a $(2 \times 2)$ max-pooling layer is used to reduce the number of parameters. This is followed by 2 (two) more convolutional layers with 128 units/filters and the same max-pooling process. There are then 2 more convolutional layers with 256 units/filters with max-pooling and, finally, 4 more convolutional layers with 512 units/filters. All of the convolutional layers were given individual batch normalization layers to reduce the over-fitting of the altered proposed VGGnet [35]. The output of the last convolutional layer is then passed through 2 (two) fully connected (FC) layers or dense layers. This is followed by an output layer of 29 nodes with Softmax output for gesture classification. This model has a total of 11 hidden layers and the input dimension of $(224, 224)$ makes this DCNN model capable of processing a total of 22,506,781 learnable parameters.

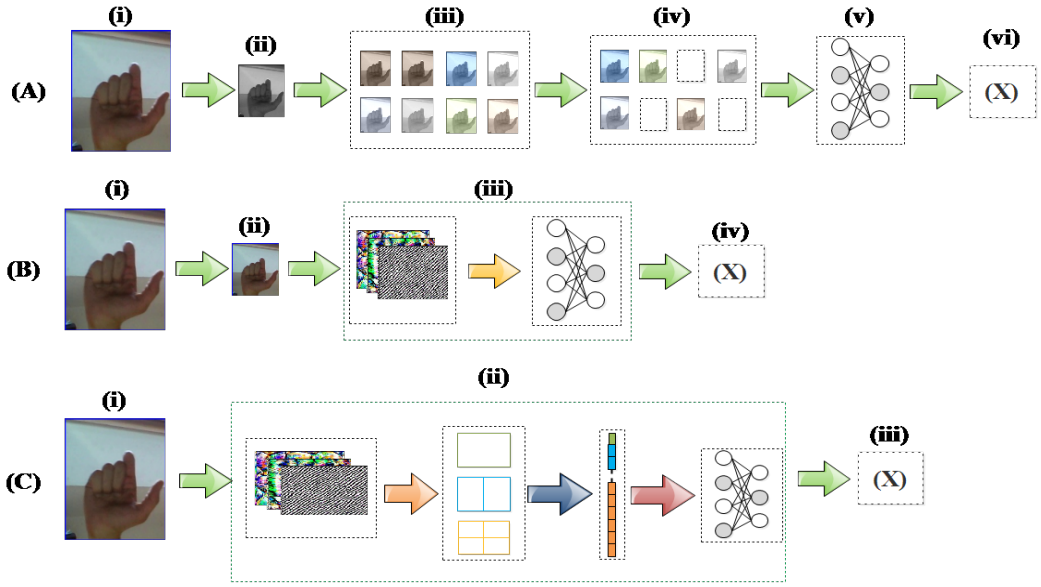


**Figure 5.** Step-by-step comparison of classification methods. (**A**) Traditional methods rely on image resizing and RGB to gray-scale conversion prior to classification. (**B**) in most DCNN-based methods, the input image needs to be resized. However, (**C**) the SPP layer-based DCNN has no need for image resizing.

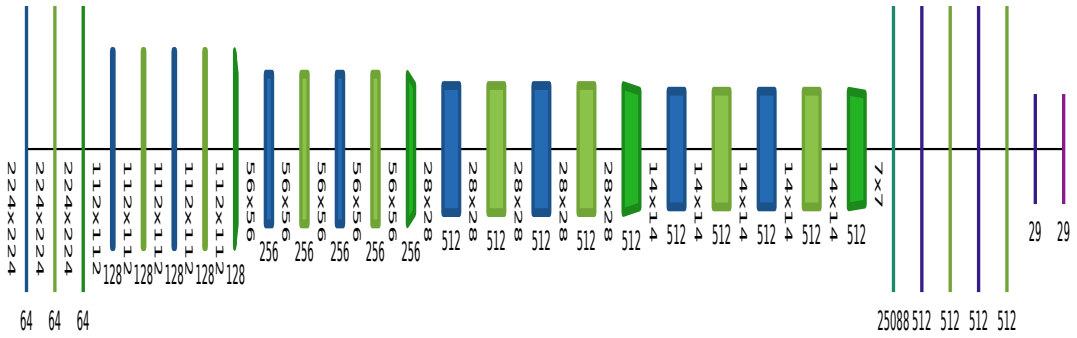


**Figure 6.** Proposed deep convolutional network visualization. The color-coded boxes represent the layers in the model. Convolutional layers are blue (■), batch normalization layers are lime (■) and max-pooling layers are forest green (■); these are followed by a flatten layer, shown in emerald green (■), a fully connected layer in violet (■) and the Softmax output in magenta (■). The number of dimensions in each layer is shown beside/below that layer.

The pruning strategy for the DCNN only works in the hidden layers. Neural network node pruning is a basic process, in which nodes are raked in order to prune them. Hu et al. [36] explored sparsity inactivation for network pruning. The exponential linear unit (ReLU) [37] activation function imposes sparsity during inference, and the average percentage of positive activation in the output can determine the importance of the neuron. L1 normalization is useful for the estimation of the saliency of feature maps in a given layer. This idea can be used to rank the filters in each layer. The trained neural network is then pruned to make it more compact in size while retaining accuracy to produce the correct results with less computational overhead. The proposed pruning of the new DCNN takes place in 2 steps. First, the model is trained with the proper dataset. Subsequently, the trained model undergoes a comparison based on validation accuracy to establish the baseline. Later model pruning is done based on the filter and nodes $l1$-normalized rankings. The pruned compact network is then retrained to give the same or higher accuracy with a reduced computational workload. As the original layer in the proposed model was extremely large, 50% pruning of convolutional layers and 25% pruning of FC layers were proposed to make the final compact DCNN that is shown in Figure 7.

Figure 7 displays a final pruned model after 50% pruning of convolutional layers and 25% pruning of FC layers. Overall, pruning resulted in more than a 50% reduction in parameters. The first convolutional layers with 64 units/filters in Figure 6 now have 32 filters, as shown in Figure 7. The same strategy was implemented in the next two convolutional layers, which have 128 units/filters in Figure 6. The pruned network has 2 convolutional layers with 64 units/filters after the first 32 node/filter layer, as shown in Figure 7. The pruning of the next two convolutional layers with 256 units/filters and the last four convolutional layers with 512 units/filters in Figure 6 followed, resulting in 2 convolutional layers with 128 units/filters and the last 4 convolutional layers with 128 units/filters, as shown in Figure 7. The FC layers of the original model each had 512 nodes, so 25% pruning of both layers leaves 334 nodes in each hidden FC layer, as displayed in Figure 7. Table 1 shows the exact number of parameters in each layer in the original and pruned network.

**Table 1.** Parameters in the proposed deep convolutional network and pruned deep convolutional network.

| **Original Deep Convolutional Network** | | | **Pruned Deep Convolutional Network** | | |
|---|---|---|---|---|---|
| **Layers** | **Output Shape** | **Parameters** | **Layers** | **Output Shape** | **Parameters** |
| Input | 224, 224, 3 | 0 | Input | 224, 224, 3 | 0 |
| (Conv2D) 64 | 224, 224, 64 | 1792 | (Conv2D) 32 | 224, 224, 32 | 896 |
| Batch Normalization | 224, 224, 64 | 256 | Batch Normalization | 224, 224, 32 | 128 |
| Max Pooling (2 × 2) | 112, 112, 64 | 0 | Max Pooling (2 × 2) | 112, 112, 32 | 0 |
| (Conv2D) 128 | 112, 112, 128 | 73,856 | (Conv2D) 64 | 112, 112, 64 | 18,496 |
| Batch Normalization | 112, 112, 128 | 512 | Batch Normalization | 112, 112, 64 | 256 |
| (Conv2D) 128 | 112, 112, 128 | 147,584 | (Conv2D) 64 | 112, 112, 64 | 36,928 |
| Batch Normalization | 112, 112, 128 | 512 | Batch Normalization | 112, 112, 64 | 256 |
| Max Pooling (2 × 2) | 56, 56, 128 | 0 | Max Pooling (2 × 2) | 56, 56, 64 | 0 |
| (Conv2D) 256 | 56, 56, 256 | 295,168 | (Conv2D) 128 | 56, 56, 128 | 73,856 |
| Batch Normalization | 56, 56, 256 | 1024 | Batch Normalization | 56, 56, 128 | 512 |
| (Conv2D) 256 | 56, 56, 256 | 590,080 | (Conv2D) 128 | 56, 56, 128 | 147,584 |
| Batch Normalization | 56, 56, 256 | 1024 | Batch Normalization | 56, 56, 128 | 512 |
| Max Pooling (2 × 2) | 56, 56, 256 | 0 | Max Pooling (2 × 2) | 28, 28, 128 | 0 |
| (Conv2D) 512 | 28, 28, 512 | 1,180,160 | (Conv2D) 256 | 28, 28, 256 | 295,168 |
| Batch Normalization | 28, 28, 512 | 2048 | Batch Normalization | 28, 28, 256 | 1024 |
| (Conv2D) 512 | 28, 28, 512 | 2,359,808 | (Conv2D) 256 | 28, 28, 256 | 590,080 |
| Batch Normalization | 28, 28, 512 | 2048 | Batch Normalization | 28, 28, 256 | 1024 |
| Max Pooling (2 × 2) | 14, 14, 512 | 0 | Max Pooling (2 × 2) | 14, 14, 256 | 0 |
| (Conv2D) 512 | 14, 14, 512 | 2,359,808 | (Conv2D) 256 | 14, 14, 256 | 590,080 |
| Batch Normalization | 14, 14, 512 | 2048 | Batch Normalization | 14, 14, 256 | 1024 |
| (Conv2D) 512 | 14, 14, 512 | 2,359,808 | (Conv2D) 256 | 14, 14, 256 | 590,080 |
| Batch Normalization | 14, 14, 512 | 2048 | Batch Normalization | 14, 14, 256 | 1024 |
| Max Pooling (2 × 2) | 7, 7, 512 | 0 | Max Pooling (2 × 2) | 7, 7, 256 | 0 |
| Flatten | 25,088 | 0 | Flatten | 12,544 | 0 |
| (Fully Connected) 512 | 512 | 12,845,568 | (Fully Connected) 384 | 384 | 4,817,280 |
| Batch Normalization | 512 | 2048 | Batch Normalization | 384 | 1536 |
| (Fully Connected) 512 | 512 | 262,656 | (Fully Connected) 384 | 384 | 147,840 |
| Batch Normalization | 512 | 2048 | Batch Normalization | 384 | 1536 |
| (Fully Connected) 512 | 29 | 14,877 | (Fully Connected) 29 | 29 | 11,165 |
| Output Softmax | 29 | 0 | Output Softmax | 29 | 0 |
| Total Parameters | | 22,506,781 | Total Parameters | | 7,328,285 |

The original proposed model presented in Figure 6 has a final trainable parameters, or the weight value holder of 22,506,781 as compared to 7,328,285 parameters in Figure 7 as noted in the Table 1. The proposed 50% pruning of the convolutional nodes also results in half of the parameters in the pruned model. Batch Normalization parameters also cut in half as a result of the parameter reduction or pruning. The output of the convolutional layers is flattened and feed into the FC layers as a design of DCNN. Table 1 demonstrates a sharp reduction of 4,817,280 parameters of first hidden FC as oppose to 12,845,568 parameters in the original model in Figure 6. This data computed based on pruning shows potential for pruning the DCNN at a high rate in each hidden layer consists of convolution and dense/fully connected nodes. However, it is not possible to prune the last FC layer in the DCNN by design due to the output node fixation of DCNN. This is why the Table 1 shows no change in the last FC later with 29 nodes, as those are the classes/labels for the dataset.

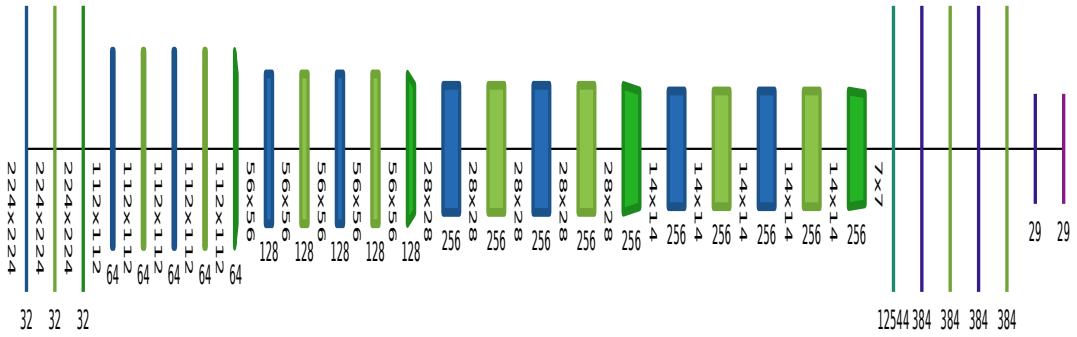


**Figure 7.** Deep convolutional network after pruning; The color-coded boxes represent the layers in the model. Convolutional layers are blue (■), batch normalization layers are lime (■) and max-pooling layers are forest green (■); these are followed by a flatten layer, shown in emerald green (■), a fully connected layer in violet (■) and the Softmax output in magenta (■). The number of dimensions in each layer is shown beside/below that layer.

### *4.2. Integration of Multi-Spatial Pyramid Pooling into the Pruned DCNN*

The model and the pruning strategy described in the section above had a fundamental limitation when it comes to the image input dimension. This results in a forcible data resizing and, thus, accuracy reduction, as stated in Figure 3. He et al. [4] proposed SPP in DCNN improves the accuracy and get rid of the input dimensions restriction. Now, some research [38,39] have suggested combining various spatial pooling combinations including the original research from He et al. [4]. Now, adopting the DCNN to fit in any arbitrary image input dimensions, we proposed a 3 multi-scale based pooling that has three $(1 \times 1)$, $(2 \times 2)$ and $(4 \times 4)$ multi-scale SPP. Table 2 shows the modified pruned DCNN in the Figure 7 with the proposed Multi-scale SPP fitted after the convolutions.

**Table 2.** Proposed spatial pyramid pooling in the deep convolutional network. Input (X, X) denotes any arbitrary input image dimensions.

| Layers | Output Shape | Parameters |
|---|---|---|
| Input | X, X, 3 | 0 |
| (Conv2D) 32 | X, X, 32 | 896 |
| Batch Normalization | X, X, 32 | 128 |
| Max Pooling $(2 \times 2)$ | X, X, 32 | 0 |
| (Conv2D) 64 | X, X,6 4 | 18,496 |
| Batch Normalization | X, X, 64 | 256 |
| (Conv2D) 64 | X, X, 64 | 36,928 |
| Batch Normalization | X, X, 64 | 256 |
| Max Pooling $(2 \times 2)$ | X, X, 64 | 0 |
| (Conv2D) 128 | X, X, 128 | 73,856 |
| Batch Normalization | X, X, 128 | 512 |
| (Conv2D) 128 | X, X, 128 | 147,584 |
| Batch Normalization | X, X, 128 | 512 |
| Max Pooling $(2 \times 2)$ | X, X, 128 | 0 |
| (Conv2D) 256 | X, X, 256 | 295,168 |
| Batch Normalization | X, X, 256 | 1024 |
| (Conv2D) 256 | X, X, 256 | 590,080 |
| Batch Normalization | X, X, 256 | 1024 |
| Max Pooling $(2 \times 2)$ | X, X, 256 | 0 |
| (Conv2D) 256 | X, X, 256 | 590,080 |
| Batch Normalization | X, X, 256 | 1024 |
| (Conv2D) 256 | X, X, 256 | 590,080 |
| Batch Normalization | X, X, 256 | 1024 |
| Max Pooling $(2 \times 2)$ | X, X, 256 | 0 |
| Spatial Pyramid Pooling | 5376 | 0 |
| (Fully Connected) 384 | 384 | 2,064,384 |
| Batch Normalization | 384 | 1536 |
| (Fully Connected) 384 | 384 | 147,840 |
| Batch Normalization | 384 | 1536 |
| (Fully Connected) 29 | 29 | 11,165 |
| Output Softmax | 29 | 0 |
| Total Parameters | | 4,575,389 |

As the last convolutional layer has 256 filters the SPP has 265 convolutional feature input. Subsequently, the data are filtered through 3 separate $(1 \times 1)$, $(2 \times 2)$, and $(4 \times 4)$ multi-scaled SPP merged together. This makes the total parameters of SPP layer $(256 \times 1 \times 2 \times 4) = 5376$ parameters. Now, the number of multi-layer SPP $(1, 2, 4)$ has been decided by both trial and error and suggestion from other researches [38,39]. Although increasing the number of pooling might further increase the accuracy, but it will add more computation overhead to the DCNN model. This proposed pruned DCNN has already been optimized with less than half of the original weight parameter with SPP based pooling. Introducing multi-scale SPP gets rid of this dimension restriction and improves the accuracy.

### *4.3. Practical Classification Algorithm for Real-Time Gesture-To-Word Decoding*

The proposed SPP based Compact DCNN will give proper results for hand gestures in real-time. However, to the best of our knowledge, a practical video to gesture decoding algorithm with SPP based DCNN has not yet been properly researched. In this subsection, we have devised a novel practical approach to decode the hand gesture to the consecutive algorithm in real-time. Algorithm 1 has shown the approach for decoding gesture(s) video in real-time.

**Algorithm 1** Hand Gesture Decoding Algorithm

**Input:** Hand gesture video or image frames set
**Output:** Decoded alphabets instruction set
***BEGIN***
**Step 1:** Process the video for the given frames per second (fps)
**Step 2:** Transform the images as 2D array $(M, N)$ where $M$ is fps and the N will be the instruction/word length (images)
**Step 3:** Apply the Proposed Compact Spatial Pyramid Pooling Deep Convolutional Neural Network to transform each images
**Step 4:** Column wise Majority Selection
**Step 5:** Return array of $n$-sized output instruction
***END***

The ASL sign language has 29 distinct classes denoting all of the English alphabet A to Z, along with *space*, *del*, and *nothing* characters gesture for additional support. The gestures set can be mathematically computed as a permutation problem with Equation (5),

$$P(n, r) = \frac{n!}{(n-r)!} \tag{5}$$

Here, $n$ is the total class of 29 gestures and $r$ is the selected sets. Giving the input into Equation (5) gives a total of 570,024 usable gestures for various instructions. Another important step of the gesture is the frame per second (fps) processing. The modern video camera has the standard of 30 fps in real life. Accordingly, the input video containing a 4 instruction set will have 120 frames to decode. It transforms the images into a $2D$ array of $(M, N)$ dimensions and then applies the proposed compact SPP-DCNN to transform each image into a corresponding gesture alphabet label. At this part of the algorithm, the majority member selection was done based on the median operator applied in the predicted array, as shown in Equation (6)

$$\text{Majority}(x_1, x_2, \ldots, x_n) = \left\lfloor \frac{1}{2} + \frac{(\sum_{i=1}^{n} x_i) - 1/2}{n} \right\rfloor \tag{6}$$

Here, $x_1, x_2, \ldots, x_n$ denotes the decoded label in each form of the array. Now, as a simple example of 5 fps gesture of $\langle A, B, C, D \rangle$ can produce $[0, 0, 0, 1]$ for the first row of data with the SPP-DCNN with 95% accuracy. However, the proposed majority member selection will change the result into the correct gesture based on the majority member in that row. This practical approach makes this proposed algorithm extremely effective for practical gesture decoding. A detailed accuracy analysis from this proposed algorithm has been discussed in Section 7 for more clarification.

## 5. Dataset Description

DCNN is practically useless without training and validating with proper data. In our research, we aim to develop practical hand gesture decoding in real-time. A proper dataset with enough label would ensure the validity of the proposed method. American Sign Language (ASL) is a generalized gesture set that has been curated and standardised by National Institute of Deafness and Other Communication Disorders (NIDCD). The precise origins of ASL are not clear, although some claim that it originated first from combining of local hand gestures and French sign language (LSF, or Langue des Signes Française) more than two centuries ago. There are a lot of datasets that curate a different form of ASL [26], especially the only one focused on the numerical hand gestures [11,40]. So, to be more robust with more practical application, this research focused on the data that were developed in Kaggle ASL challenge [28].

The data gathering for the training comprises 87,000 images which are $(200 \times 200)$ dimensions. There are 29 categories, including 26 for the characters A–Z and 3 for SPACE, DELETE, and EMPTY. These 3 sections are a huge benefit in implementations and decoding gestures in real-time. The test data collection includes only 29 images categories, to allow for real-world reference images to be included. Figure 8 displays the whole data class examples. Although, Islam et al. [26] proposed and collected dataset containing 26 classes of same ASL. However, the prepossessing from RGB to grayscale makes it not usable for our learning purpose. Kaggle ASL challenge dataset was more suitable as it has the dataset with no prepossessing done to it prior to classification. It is worth mentioning that although this dataset titled as the ASL hand gesture dataset, for ASL finger spelling same principle applies for labeling. This makes them virtually the same task with different terminologies.


**Figure 8.** English ASL alphabet based hand gesture dataset. Here, the first 26 (Twenty six) gesture represents from A to Z; 3 extra gesture classes for SPACE, DELETE, and EMPTY.

Table 3 shows the training testing split of the dataset for proper validation. The splitting was done randomly to ensure bias-free training and data augmentation, random rotation and shifting were used during training for ensuring reduced over-fitting.

**Table 3.** Training/testing separation statistics.

| Total Data | Training Data | Testing Data |
|---|---|---|
| 87,000 | 69,600 | 17,400 |

## 6. Evaluation Metrics

It is important to establish proper evaluation metrics for analyzing the performance of any machine learning algorithm. Although, most of the DCNN related research solely focuses on accuracy for validation. However, precision, recall, and $f1$-score can also help to interpret the result more accurately. Precision $P$, Recall $R$, and $f1$ Score can be calculated, as follows,

$$P = TP/(TP + FP) \tag{7}$$

$$R = TP/(TP + FN) \tag{8}$$

$$f1 = 2PR/(P + R) \tag{9}$$

Here $TP$ is the number of true positives, i.e., the number of correctly classified faulty instances. $FP$ is the number of false positives, i.e., the number of normal instances that were wrongfully classified as faulty ones. $FN$ stands for false negatives, which is the number of faulty instances classified as normal ones. the properly labeled cases were denoted as True Negative $TN$. Additionally, the accuracy (AC) is also computed, as follows,

$$AC = (TP + TN)/(TP + TN + FP + FN) \tag{10}$$

Additionally, $R^2$ (coefficient of determination) regression score function has been computed for 5 consecutive testing of all 3 proposed model for evaluation. Assuming the original label as $y$ and predicted labels as $\hat{y}$, the $R^2$ will be

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{11}$$

Here, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \epsilon_i^2$. It reflects the proportion of variance that the independent variables in the model have defined. It gives an indicator of fitness effectiveness and, thus, a measure of how well the model is able to assess unknown samples. For the newly proposed member majority function, the mean majority is calculated, as follows,

$$Mj_{mean}(a_1, a_2, \ldots, a_n) = 100 * \frac{1}{n}\sum_{i=1}^{n} a_i * k \tag{12}$$

$a_i * k$ is the corrected majority derived from the dataset. Basically, it is a mean average of the majority count in the array $[a_1, a_2, \ldots, a_n]$. The detailed analysis of the result evaluated based on this metric is discussed in Section 7.

## 7. Results

The implementation and both pieces of training with validation and testing of the proposed DCNN network were exclusively done by using the Python programming language. Python allows for creation of a virtual environment and a cloning mechanism in that environment with relative ease. Thus, developing the proposed DCNN algorithm with SPP in a Python-based application allows for rapid prototyping and instant deployment in a variety of practical scenarios. Experiments of proposed

DCNN node pruning and SPP integration were constructed and modified by Python-based Keras library, which has a TensorFlow library as a backend system [41]. A high-performance server computer with Intel Corporation Xeon E5/Core i7, 32 GB of RAM, and Nvidia GPU RTX 2080 with Ubuntu 16.04.6 LTS Operating System (OS) was used for training and application deployment. The DCNN training parameters are 100 epochs each, with a 40 batch size, the learning rate was 0.001 for faster convergence. Convolution and Dense or FC layers used in these proposed DCNNs were both built with Relu activation [37], along with Adadelta Optimization and Categorical Crossentropy for loss/error function. The main addition in this ensemble is the introduction of Batch normalization [42] to reduce over-fitting and the convergence of data during training. The original VGG-11 was not utilized with batch normalization. The original and both pruned and SPP with pruned DCNN was trained and tested using a single Nvidia RTX 2080 GPU instance. However, in modern application deployment, it is necessary to use parallel GPU training and deployment. Moreover, sometimes, more than two or three instances of GPUs are available to use in real-time application. In order to deploy with GPU based parallelism, the final proposed SPP based DCNN model has trained with 2 Nvidia RTX 2080 GPU instances.

In Figures 9 and 10, loss chart displays the gradual error reduction during training and validation/testing. Even though the loss chart is comparatively smooth, the accuracy varies drastically during training and validation. This is due to the data augmentation and highly randomised testing in each iteration. This forces the model to learn without over-fitting. They also display the training and validation of the Original Proposed DCNN and retraining after the pruning was applied. The original and the pruned training were both done for 100 iterations of Epoch. Now, both Figures 9 and 10 have shown instability during the training and this phenomenon can be described by the random data augmentation that is done to reduce over-fitting. However, model weights were saved at the end of every epoch to select the best weigh from the training. The DCNN had the highest accuracy of 90.0% and the pruned DCNN has 91.0% accuracy. This finding is initially surprising, as all the previous literature suggested that retaining or slight decrease in accuracy usually occurs after the pruning. However, this occurrence can also be explained by learning data volume and DCNN size. The first proposed modified VGG-11, like DCNN, had 11 total hidden layers and most of the last convolution layer along with the first hidden FC layer of the neural network has a 12,845,568 and 2,359,808 weight parameters. On the other hand, the total training instances were 69,600 images. A simple representative convolution of six parameters each can be "memorized" or saved into the weight parameter and, thus, an over-fitting occurs. Now, this result actually proves the hypothesis at the beginning of this research. The hand gesture classifier has less data than the Image-net Challenge dataset and does not need a bigger network, like the proposed original, to achieve higher accuracy. Additionally, this theory can be proved more in the analysis of the weight and the $l1$-normalization ranking of the nodes and filters in all hidden layers.

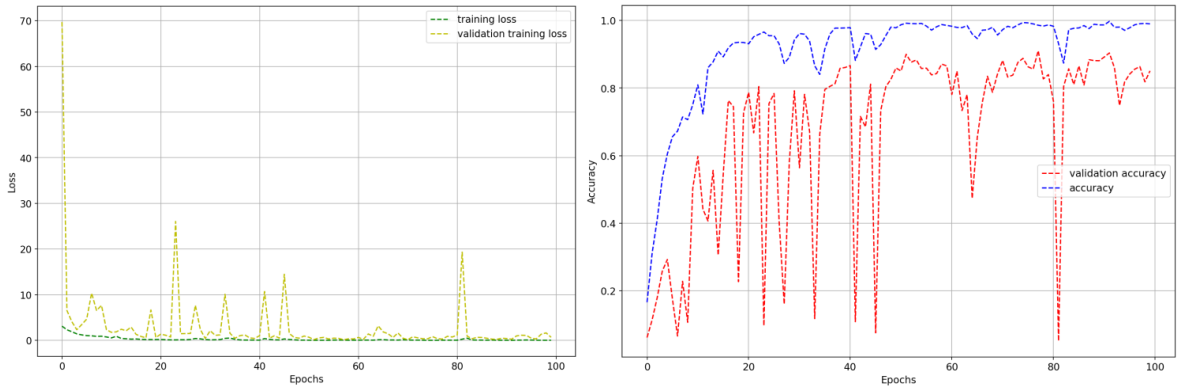


**Figure 9.** Training/testing loss and accuracy of the proposed deep convolutional neural network.

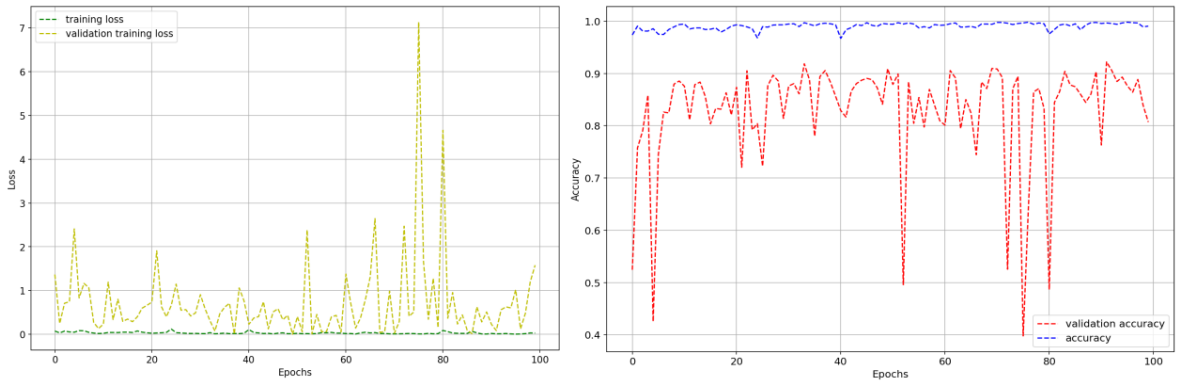


**Figure 10.** Training/testing loss and accuracy of the proposed pruned deep convolutional neural network.

Table 4 has the experimental results to prove the assumptions above. The whole worst scored 20 filters and node weights after can been seen in Table 4. The first hidden layers have different nodes. such as middle nodes as less weight, but the last convolutional layer almost exclusively has low scoring weight outermost filters when compared to the projection of previous convoluted output.

**Table 4.** Node pruning details in each layer. The worst scored nodes can be used in future references as a reproducibility.

| Layers | Numbers of Original Node/Filters | Numbers of Pruned Node/Filters | Percentage of Pruning | Worst Scored 20 Filters/Nodes |
|---|---|---|---|---|
| (Conv2D) 64 | 64 | 32 | 50 | 29, 7, 38, 31, 23, 54, 27, 40, 6, 42, 45, 61, 44, 25, 20, 15, 49, 4, 50, 1 |
| (Conv2D) 128 | 128 | 64 | 50 | 70, 39, 1, 64, 60, 110, 116, 119, 50, 84, 18, 107, 42, 89, 48, 15, 85, 7, 12, 58 |
| (Conv2D) 128 | 128 | 64 | 50 | 16, 49, 101, 100, 36, 88, 123, 91, 48, 97, 95, 78, 23, 55, 93, 68, 74, 108, 86, 82 |
| (Conv2D) 256 | 256 | 128 | 50 | 250, 82, 230, 186, 121, 62, 228, 35, 199, 64, 17, 133, 60, 143, 58, 57, 139, 31, 255, 135 |
| (Conv2D) 256 | 256 | 128 | 50 | 100, 88, 196, 245, 49, 118, 251, 170, 42, 138, 107, 92, 160, 238, 143, 199, 253, 191, 233, 36 |
| (Conv2D) 512 | 512 | 256 | 50 | 246, 273, 125, 166, 396, 287, 9, 233, 59, 111, 483, 22, 70, 423, 27, 370, 469, 232, 7, 372 |
| (Conv2D) 512 | 512 | 256 | 50 | 58, 119, 497, 221, 482, 487, 253, 251, 267, 160, 296, 204, 23, 179, 214, 278, 114, 48, 76, 414 |
| (Conv2D) 512 | 512 | 256 | 50 | 369, 454, 213, 317, 385, 395, 134, 147, 160, 346, 251, 58, 124, 360, 45, 205, 352, 445, 33, 498 |
| (Conv2D) 512 | 512 | 256 | 50 | 29, 79, 90, 313, 96, 27, 246, 436, 373, 298, 255, 148, 229, 262, 360, 9, 264, 150, 131, 172 |
| (Fully Connected) 512 | 512 | 384 | 25 | 251, 309, 244, 375, 218, 478, 153, 189, 146, 128, 357, 104, 325, 463, 430, 394, 253, 27, 441, 203 |
| (Fully Connected) 512 | 512 | 384 | 25 | 454, 98, 250, 410, 438, 124, 332, 175, 434, 297, 360, 5, 505, 436, 510, 69, 287, 32, 166, 504 |

Figure 11 is the visual representation pruning justification in the proposed method. Original proposed DCNN's first six convolutional layers filters are shown based on the $l1$-normalized values. Half of these nodes have the normalized weight value that falls less than the median. Accordingly, pruning half of the nodes in reality did not affect the total accuracy of the proposed DCNN. Moreover, retraining the pruned model forces the weights to generalized of features detection more than "memorization" of the dataset. As a result, the pruned DCNN gained more accuracy than the original model. This practical pruning also proves that the bigger DCNN is often not useful for smaller features/labels dataset classification.

Table 5 shows the testing accuracy, precision, recall, and $f1$ score for the all 3 proposed model. Although the overall accuracy of the original DCNN had 90% over all classes, the individual precision was skewed to some classes and less in some classes. This results in bias classification accuracy in real life. However, pruning and retraining of the model made the DCNN more generalized, and the class-wise precision and recall improved significantly. The overall accuracy also increases due to the newly pruned DCNN. After pruning and retraining for 100 epochs, the general accuracy improved into 92% in the pruned DCNN. Additionally, the introduction of SPP in the pruned DCNN further improves the accuracy of up to 95%. This is around 2% improvement of the previous model, as suggested by He et al. [4]. Moreover, SPP makes the final model dimension invariant. As a result, the input image can be fed directly without any pre-processing.

**Table 5.** Result Analysis of all the Proposed Models

| | Original Model | | | | Pruned Model | | | | Convolutional Spatial Pyramid Pooling Model | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Class** | **Precision** | **Recall** | ***f*1-Score** | **Support** | **Precision** | **Recall** | ***f*1-Score** | **Support** | **Precision** | **Recall** | ***f*1-Score** | **Support** |
| 0 | 0.95 | 0.97 | 0.96 | 146 | 0.96 | 0.95 | 0.96 | 137 | 0.9 | 1 | 0.95 | 140 |
| 1 | 0.93 | 1 | 0.96 | 137 | 0.97 | 0.94 | 0.96 | 125 | 0.97 | 1 | 0.99 | 151 |
| 2 | 0.97 | 1 | 0.98 | 139 | 0.99 | 0.98 | 0.99 | 131 | 1 | 0.98 | 0.99 | 150 |
| 3 | 1 | 0.89 | 0.94 | 142 | 1 | 0.77 | 0.87 | 154 | 1 | 1 | 1 | 119 |
| 4 | 0.83 | 0.98 | 0.9 | 140 | 0.77 | 0.99 | 0.87 | 133 | 0.91 | 0.89 | 0.9 | 141 |
| 5 | 1 | 0.95 | 0.98 | 133 | 0.97 | 1 | 0.99 | 132 | 1 | 0.97 | 0.98 | 146 |
| 6 | 0.97 | 0.89 | 0.92 | 132 | 1 | 0.89 | 0.94 | 152 | 1 | 0.93 | 0.97 | 136 |
| 7 | 0.95 | 0.94 | 0.95 | 143 | 0.95 | 0.99 | 0.97 | 141 | 0.94 | 0.98 | 0.96 | 151 |
| 8 | 1 | 0.77 | 0.87 | 138 | 1 | 0.67 | 0.8 | 170 | 0.98 | 0.91 | 0.94 | 132 |
| 9 | 0.97 | 0.94 | 0.96 | 136 | 0.83 | 0.96 | 0.89 | 142 | 0.96 | 0.96 | 0.96 | 146 |
| 10 | 0.99 | 1 | 1 | 141 | 0.95 | 0.94 | 0.94 | 127 | 1 | 1 | 1 | 127 |
| 11 | 0.99 | 0.99 | 0.99 | 141 | 1 | 0.98 | 0.99 | 134 | 1 | 1 | 1 | 143 |
| 12 | 0.82 | 0.76 | 0.79 | 148 | 0.87 | 0.84 | 0.85 | 154 | 0.73 | 0.98 | 0.83 | 113 |
| 13 | 0.56 | 0.55 | 0.56 | 139 | 0.8 | 0.68 | 0.73 | 142 | 0.98 | 0.6 | 0.75 | 134 |
| 14 | 0.8 | 0.85 | 0.82 | 141 | 0.73 | 0.87 | 0.79 | 112 | 0.96 | 0.89 | 0.93 | 133 |
| 15 | 0.94 | 1 | 0.97 | 114 | 0.95 | 1 | 0.97 | 133 | 0.98 | 1 | 0.99 | 131 |
| 16 | 0.93 | 0.99 | 0.96 | 139 | 0.95 | 0.95 | 0.95 | 146 | 0.99 | 0.98 | 0.99 | 135 |
| 17 | 0.99 | 0.87 | 0.93 | 124 | 0.94 | 0.96 | 0.95 | 138 | 0.98 | 0.97 | 0.98 | 143 |
| 18 | 0.69 | 0.64 | 0.66 | 135 | 0.66 | 0.97 | 0.78 | 132 | 0.68 | 0.94 | 0.79 | 144 |
| 19 | 0.95 | 0.75 | 0.84 | 129 | 0.97 | 0.81 | 0.89 | 143 | 0.89 | 0.96 | 0.92 | 140 |
| 20 | 0.78 | 0.99 | 0.88 | 143 | 0.88 | 0.91 | 0.89 | 144 | 1 | 0.97 | 0.98 | 150 |
| 21 | 0.88 | 0.99 | 0.93 | 137 | 0.92 | 0.98 | 0.95 | 126 | 0.98 | 0.99 | 0.99 | 128 |
| 22 | 0.84 | 0.95 | 0.89 | 140 | 0.83 | 0.99 | 0.91 | 130 | 0.95 | 1 | 0.97 | 124 |
| 23 | 0.7 | 0.57 | 0.63 | 128 | 0.99 | 0.6 | 0.75 | 142 | 0.97 | 0.44 | 0.6 | 133 |
| 24 | 0.87 | 0.91 | 0.89 | 137 | 0.92 | 0.92 | 0.92 | 130 | 0.96 | 0.9 | 0.93 | 144 |
| 25 | 0.88 | 0.97 | 0.92 | 144 | 0.94 | 0.97 | 0.95 | 118 | 0.85 | 1 | 0.92 | 144 |
| 26 | 0.96 | 0.91 | 0.93 | 148 | 0.95 | 0.95 | 0.95 | 153 | 0.99 | 0.99 | 0.99 | 140 |
| 27 | 1 | 0.99 | 0.99 | 144 | 0.9 | 0.98 | 0.93 | 131 | 0.92 | 1 | 0.96 | 142 |
| 28 | 0.99 | 1 | 1 | 142 | 0.97 | 1 | 0.98 | 148 | 1 | 1 | 1 | 140 |
| accuracy | - | - | 0.9 | 4000 | - | - | 0.91 | 4000 | - | - | 0.94 | 4000 |
| macro avg. | 0.9 | 0.9 | 0.9 | 4000 | 0.92 | 0.91 | 0.91 | 4000 | 0.95 | 0.94 | 0.94 | 4000 |
| weighted avg. | 0.9 | 0.9 | 0.9 | 4000 | 0.92 | 0.91 | 0.91 | 4000 | 0.95 | 0.94 | 0.94 | 4000 |

The pruning of the DCNN not only improved the accuracy, it also compacted the DCNN with less weight parameter. Table 6 shows the final pruning statistics of the proposed DCNN. The original DCNN had a total of 22,498,973 weight parameters and the pruned DCNN had the final 7,323,869 weight parameters. This pruning makes the network more than 3× smaller than the original, with a 67.45% compression rate.

**Table 6.** Deep Convolutional Network Pruning Statistics.

| **Original Parameters** | **After Pruned** | **Pruned Parameters** |
|---|---|---|
| 22,498,973 | 7,323,869 | 15,175,104 |
| Total Compression (%) | 67.45% | |

Figure 12 displays that some common mislabeling occurs in all proposed DCNN models. In hindsight, this might contradict the high accuracy of pruned model and SPP DCNN, but further examination of the input images shows that the original input image is often too visually obscure to separate distinguishable features with convolution. Furthermore, some of the images are more visually similar to predicted class than ground truth.

Table 7 shows the advanced performance of the all 3 proposed DCNN. Each model's performance is based on a 4000 random sample input and initialized 5 times for randomness in each epoch. Accuracy and $R^2$ score for all 3 model both prove the viability of using this proposed DCNN in real-life scenarios. A comparison among 3 models based on the $R^2$-score makes the SPP-based DCNN the best model to use in real-life applications. This ensures the prediction quality of the proposed SPP-DCNN suitable for a faster reliable hand gesture classifier. The proposed gesture decoder algorithm needs faster prediction output to apply in real-time in the gesture decoder systems. Table 8 shows a comparative run-time of the proposed algorithms in real-time based on the generated data from Algorithm 1. The setup was followed, as described at the beginning of this Section 7. Now, the proposed 3 models perform relatively well. However, it is noticeable the run-time for a video sequence containing 4 gestures have $19.30 \pm 1.32$ s average. On the other hand, the pruned version was sped up significantly

having $9.34 \pm 1.37$ s average. This is predictable, as the pruned model has less weight parameters to calculate in real-time. Moreover, in real-life, some of the server-based application has the option for multi-GPU based parallelization. The proposed SPP-based DCNN was also implemented as a multi-GPU based in this experiment and had an astonishing $0.013 \pm 0.019$ s average run when compared to the other models.

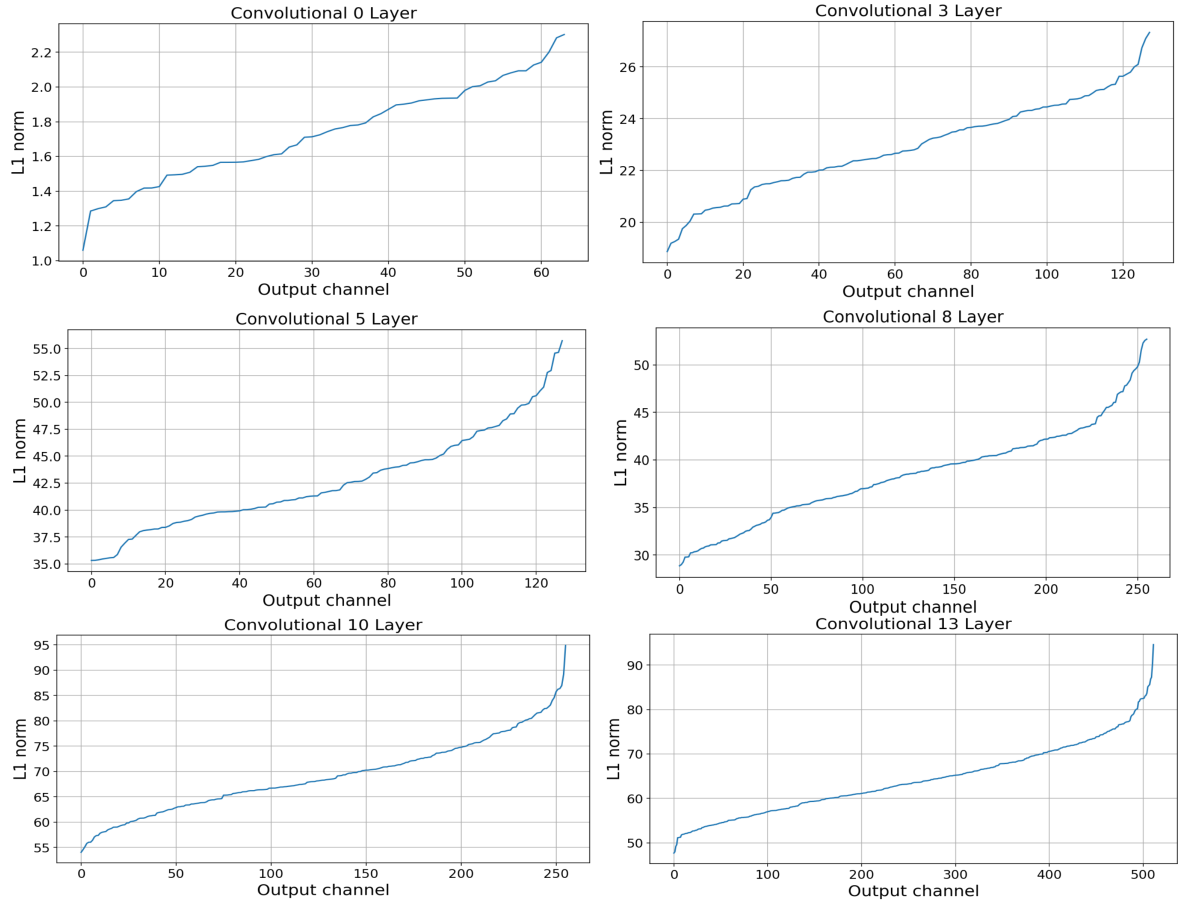


**Figure 11.** Filters of the first six layers of the proposed convolutional neural network, ranked by weight l1 normalization.


**Figure 12.** Examples of mis-labeling by all three proposed models; (**A**) original model; (**B**) pruned model; and, (**C**) spatial pyramid pooling model

**Table 7.** Advanced performance of the proposed deep convolutional neural network models. Each model's performance is based on 4000 random sample input; initialized 5 times for randomness in each epoch.

| Original Model | | Pruned Model | | Spatial Pyramid Pooling Model | |
|---|---|---|---|---|---|
| **Accuracy** | **$R^2$-Score** | **Accuracy** | **$R^2$-Score** | **Accuracy** | **$R^2$-Score** |
| 90.9761 | 0.9050 | 91.7250 | 0.9262 | 93.1249 | 0.9415 |
| 90.9762 | 0.9022 | 91.3250 | 0.9206 | 94.3750 | 0.9599 |
| 90.9761 | 0.9125 | 91.2750 | 0.9392 | 93.8000 | 0.9384 |
| 90.9761 | 0.9227 | 92.6429 | 0.9206 | 93.3571 | 0.9487 |
| 90.5814 | 0.9148 | 89.1129 | 0.9241 | 93.1500 | 0.9552 |

**Table 8.** Deep convolutional network classification in real-time (mean $\pm$ standard deviation of several runs, 10 loops each); containing four sequence characters/finger-spelling.

| Model Name | Run Time (s) | Comment |
|---|---|---|
| Modified VGG-like Proposed Model | $19.30 \pm 1.32$ | - |
| Proposed Pruned Model | $9.34 \pm 1.37$ | - |
| Spatial Pyramid Pooling Model | $4.10 \pm 8.97$ | - |
| Spatial Pyramid Pooling Model | $0.013 \pm 0.019$ | (Multi-GPU) |

Table 9 shows the results of multi-GPU based parallelization in greater detail. Here, the 6 gesture sequence video or 6 consecutive alphabets finger-spelling takes an average of $267 \pm 2.13$ ms to decode per frames. Based on these results, the proposed algorithm is fast enough to decode gestures in the real world.

**Table 9.** Compact deep spatial pyramid pooling convolutional network classification in real-time (mean $\pm$ standard deviation of seven runs, 10 loops each) (with multi-GPU functionality; 2 GTX-2080).

| Total Class in Sequence | Run Time (ms) |
|---|---|
| 3 | $136 \pm 1.98$ |
| 4 | $173 \pm 3.36$ |
| 5 | $202 \pm 8.97$ |
| 6 | $267 \pm 2.13$ |

Table 10 shows the results of gesture decoding by the proposed algorithm while using a video with various frame rates. The mean majority accuracy was counted based on Equation (12) from Section 6. This provides an idea of the overall ability of the proposed decoding algorithm. Even at 60 fps, the decoder gives 99% accuracy. The decrease in accuracy with increased fps can be explained by the transitional frames between gestures. However, as this error is minimal, the algorithm can cope with this minuscule source of error. Based on this experiment, 30 fps is the recommended speed for fast real-world decoding for practical applications. Our proposed decoder or ASL finger-spelling system usually 100% accurate in various finger-spelling or gesture decoding. However, the mean majority in rows shows that the gradual rise of error can be cumulative and might make wrong classification in future. However, more training with diverse dataset and applying proposed algorithm in recommended 30 fps will held up the reported accuracy of any ASL-finger-spelling gesture decoding in real time.

**Table 10.** Proposed sequencer performance analysis.

| Image Samples in Each Class (fps) | Original Generated Sequence | Pruned Classifier Predicted Sequence | Mean Majority in Rows (Percent) | Spatial Pyramid Classifier Predicted Sequence | Mean Majority in Rows (Percent) |
|---|---|---|---|---|---|
| 5 | ['H' 'K' 'J' 'M'] | ['H' 'K' 'J' 'M'] | 99.20 | ['H' 'K' 'J' 'M'] | 100 |
| | ['Y' 'G' 'Z' 'X'] | ['Y' 'G' 'Z' 'X'] | 99.20 | ['Y' 'G' 'Z' 'X'] | 100 |
| | ['T' 'C' 'W' 'V'] | ['T' 'C' 'W' 'V'] | 99.20 | ['T' 'C' 'W' 'V'] | 100 |
| 10 | ['H' 'K' 'J' 'M'] | ['H' 'K' 'J' 'M'] | 99.23 | ['H' 'K' 'J' 'M'] | 100 |
| | ['Y' 'G' 'Z' 'X'] | ['Y' 'G' 'Z' 'X'] | 98.2 | ['Y' 'G' 'Z' 'X'] | 100 |
| | ['T' 'C' 'W' 'V'] | ['T' 'C' 'W' 'V'] | 98.2 | ['T' 'C' 'W' 'V'] | 99.97 |
| 30 | ['H' 'K' 'J' 'M'] | ['H' 'K' 'J' 'M'] | 99.23 | ['H' 'K' 'J' 'M'] | 99.88 |
| | ['Y' 'G' 'Z' 'X'] | ['Y' 'G' 'Z' 'X'] | 98.99 | ['Y' 'G' 'Z' 'X'] | 99.3 |
| | ['T' 'C' 'W' 'V'] | ['T' 'C' 'W' 'V'] | 98.99 | ['T' 'C' 'W' 'V'] | 99.33 |
| 60 | ['H' 'K' 'J' 'M'] | ['H' 'K' 'J' 'M'] | 98.99 | ['H' 'K' 'J' 'M'] | 99.33 |
| | ['Y' 'G' 'Z' 'X'] | ['Y' 'G' 'Z' 'X'] | 98.24 | ['Y' 'G' 'Z' 'X'] | 99.67 |
| | ['T' 'C' 'W' 'V'] | ['T' 'C' 'W' 'V'] | 98.56 | ['T' 'C' 'W' 'V'] | 99.33 |

## 8. Comparison with Other Methodologies

Modern deep learning-based hand gesture classification research has been solely focused on accuracy-based classification and regression. Depending on the image dataset, the size and computational complexity of such classification algorithms are increasing rapidly. In this study, we have introduced a new real-time approach to classical gesture decoding. Islam et al. [26] approached DCNN-based classification while using the same dataset as used in this research and reported 94% accuracy over all input data. However, that method involved a background subtraction process along with data image resizing and gray-scale conversion. Moreover, integrating a multi-support vector machine (MSVM) into the DCNN might further improve classification accuracy, but ultimately the algorithm lacks practicality for real-time applications. Tushar et al. [40] approached gesture recognition in a similar fashion as the present research, but the image resizing and limitation of gesture classes to numerical hand gestures limits its utility in real-time applications. Our previous approach to node pruning with a DCNN yielded a similar increase in prediction response time, but the image resizing restriction was still present [43].

The proposed compact SPP-based DCNN model eliminates the image resizing restriction. We utilized a large ASL dataset to create a new gesture or ASL finger-spelling decoding algorithm that works in real time. Although the proposed pruned model with the SPP layer achieved the same accuracy as the latest relevant research using this dataset, the combination with the newly proposed video-based decoding upgraded the result to a maximum of 99% accuracy in real time with very low processing time. All of the previous algorithms focused on accuracy improvement. In this research, we have combined the improved accuracy with a novel algorithm to achieve a new benchmark system that provides a practical technique for the development of future applications that are based on gesture recognition.

## 9. Conclusions

The rise of edge-based remote applications has created a demand for high-performance low-cost computing-based deep learning convolutional neural networks (DCNNs). The main idea of the DCNN is not new. However, optimization and expanding the applications of neural networks are now very demanding tasks due to the rise of edge computing and real-time response-based application deployment. Decoding hand gestures in real time requires a fast DCNN capable of interpreting variable-sized image inputs owing to the variation between cameras in modern systems. In this research, we proposed a new hand gesture classification system that can classify various hand gestures

with 94% accuracy. Moreover, we integrated a spatial pyramid pooling (SPP) layer into the proposed DCNN and used node pruning to make it less computationally resource intensive and image input dimension-invariant. Consequently, the proposed SPP-DCNN is the most reliable method of real-time gesture decoding. Moreover, we have introduced a novel algorithm for video-based gesture decoding that can process a video with any arbitrary input dimensions and variable frames per second (fps), which will decode an input gesture video into consecutive gesture classes. This proposed new and faster system can also be used as an advance ASL finger-spelling recogniser. The use of these compact SPP-DCNNs in various remote smart locations with minimal computing resources will ensure high performance with lower computing costs and better connectivity.

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

**Funding:** This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2020-2016-0-00314) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation) and also are results of a study on the supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government(MOTIE) (P0011931, The Establishment Project of Industry-University Fusion District). Finally, this project was also supported by the National Research Foundation of Korea (NRF) funded by the MSIT (NRF-2018R1A4A1025559).

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

## References

1. Corballis, M.C. From mouth to hand: Gesture, speech, and the evolution of right-handedness. *Behav. Brain Sci.* **2003**, *26*, 199–208. [CrossRef] [PubMed]
2. Liu, L.; Özsu, M.T. *Encyclopedia of Database Systems*; Springer: New York, NY, USA, 2009; Volume 6.
3. LeCun, Y.; Denker, J.S.; Solla, S.A. Optimal Brain Damage. In *Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference*; Morgan Kaufmann: San Francisco, CA, USA, 1990; pp. 598–605.
4. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [CrossRef] [PubMed]
5. Vogler, C.; Metaxas, D. Adapting hidden Markov models for ASL recognition by using three-dimensional computer vision methods. In Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, Orlando, FL, USA, 12–15 October 1997; Volume 1, pp. 156–161.
6. Vogler, C.; Metaxas, D. ASL recognition based on a coupling between HMMs and 3D motion analysis. In Proceedings of the Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271), Bombay, India, 7 January 1998; pp. 363–369.
7. Mitra, S.; Acharya, T. Gesture recognition: A survey. *IEEE Trans. Syst. Man. Cybern. Part C* **2007**, *37*, 311–324. [CrossRef]
8. Hoshino, K.; Kawabuchi, I. A humanoid robotic hand performing the sign language motions. In Proceedings of the MHS2003 International Symposium on Micromechatronics and Human Science (IEEE Cat. No. 03TH8717), Nagoya, Japan, 19–22 October 2003; pp. 89–94.
9. Karami, A.; Zanj, B.; Sarkaleh, A.K. Persian sign language (PSL) recognition using wavelet transform and neural networks. *Expert Syst. Appl.* **2011**, *38*, 2661–2667. [CrossRef]
10. Weerasekera, C.S.; Jaward, M.H.; Kamrani, N. Robust asl fingerspelling recognition using local binary patterns and geometric features. In Proceedings of the 2013 International Conference on Digital Image Computing: Techniques and Applications (DICTA), Hobart, TAS, Australia, 26–28 November 2013; pp. 1–8.
11. Bhuiyan, R.A.; Tushar, A.K.; Ashiquzzaman, A.; Shin, J.; Islam, M.R. Reduction of gesture feature dimension for improving the hand gesture recognition performance of numerical sign language. In Proceedings of the 2017 20th International Conference of Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 22–24 December 2017; pp. 1–6.

12. Oz, C.; Leu, M.C. American Sign Language word recognition with a sensory glove using artificial neural networks. *Eng. Appl. Artif. Intell.* **2011**, *24*, 1204–1213. [CrossRef]
13. Vogler, C.; Metaxas, D. A framework for recognizing the simultaneous aspects of american sign language. *Comput. Vis. Image Underst.* **2001**, *81*, 358–384. [CrossRef]
14. Ranga, V.; Yadav, N.; Garg, P. American sign language fingerspelling using hybrid discrete wavelet transform-gabor filter and convolutional neural network. *J. Eng. Sci. Technol.* **2018**, *13*, 2655–2669.
15. Jung, R.; Kornhuber, H.; Da Fonseca, J.S. Multisensory Convergence on Cortical Neurons Neuronal Effects of Visual, Acoustic and Vestibular Stimuli in the Superior Convolutions of the Cat's Cortex. *Prog. Brain Res.* **1963**, *1*, 207–240.
16. LeCun, Y.; Bengio, Y. Convolutional networks for images, speech, and time series. *Handb. Brain Theory Neural Netw.* **1995**, *3361*, 1995.
17. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems. 2012; pp. 1097–1105. Available online: http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf (accessed on 19 August 2020).
18. Kılıboz, N.Ç.; Güdükbay, U. A hand gesture recognition technique for human–computer interaction. *J. Vis. Commun. Image Represent.* **2015**, *28*, 97–104. [CrossRef]
19. Kim, H.J.; Lee, J.S.; Park, J.H. Dynamic hand gesture recognition using a CNN model with 3D receptive fields. In Proceedings of the 2008 international conference on neural networks and signal processing, Nanjing, China, 7–11 June 2008; pp. 14–19.
20. Lin, H.I.; Hsu, M.H.; Chen, W.K. Human hand gesture recognition using a convolution neural network. In Proceedings of the 2014 IEEE International Conference on Automation Science and Engineering (CASE), Taipei, Taiwan, 18–22 August 2014; pp. 1038–1043.
21. Kim, S.Y.; Han, H.G.; Kim, J.W.; Lee, S.; Kim, T.W. A hand gesture recognition sensor using reflected impulses. *IEEE Sens. J.* **2017**, *17*, 2975–2976. [CrossRef]
22. Zafrulla, Z.; Brashear, H.; Starner, T.; Hamilton, H.; Presti, P. American sign language recognition with the kinect. In Proceedings of the 13th International Conference on Multimodal Interfaces, Alicante, Spain, 14–18 November 2011; pp. 279–286.
23. Ren, Z.; Yuan, J.; Meng, J.; Zhang, Z. Robust part-based hand gesture recognition using kinect sensor. *IEEE Trans. Multimed.* **2013**, *15*, 1110–1120. [CrossRef]
24. Bantupalli, K.; Xie, Y. American sign language recognition using deep learning and computer vision. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 4896–4899.
25. Tushar, A.K.; Ashiquzzaman, A.; Afrin, A.; Islam, M. A Novel Transfer Learning Approach upon Hindi, Arabic, and Bangla Numerals using Convolutional Neural Networks. *arXiv* **2017**, arXiv:1707.08385.
26. Islam, M.R.; Mitu, U.K.; Bhuiyan, R.A.; Shin, J. Hand gesture feature extraction using deep convolutional neural network for recognizing American sign language. In Proceedings of the 2018 4th International Conference on Frontiers of Signal Processing (ICFSP), Poitiers, France, 24–27 September 2018; pp. 115–119.
27. Garcia, B.; Viesca, S.A. Real-time American sign language recognition with convolutional neural networks. *Convolutional Neural Netw. Vis. Recognit.* **2016**, *2*, 225–232.
28. Akash. ASL Alphabet Dataset. 2018. Available online: https://www.kaggle.com/grassknoted/asl-alphabet (accessed on 19 August 2020).
29. Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep Learning*; MIT Press: Cambridge, UK, 2016; Volume 1.
30. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
31. LeCun, Y. The MNIST Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 19 August 2020).
32. Krizhevsky, A.; Hinton, G. Convolutional deep belief networks on cifar-10. 2010, unpublished manuscript.
33. Sivic, J.; Zisserman, A. Video Google: A Text Retrieval Approach to Object Matching in Videos. In Proceedings of the IEEE International Conference on Computer Vision (ICCV 2003), Nice, France, 13–16 October 2003; p. 1470.
34. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.

35. Understanding the Backward Pass through Batch Normalization Layer. Available online: https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html (accessed on 6 September 2017).
36. Hu, H.; Peng, R.; Tai, Y.; Tang, C.; Trimming, N. A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv* **2016**, arXiv:1607.03250.
37. Clevert, D.A.; Unterthiner, T.; Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv* **2015**, arXiv:1511.07289.
38. Yue, J.; Mao, S.; Li, M. A deep learning framework for hyperspectral image classification using spatial pyramid pooling. *Remote Sens. Lett.* **2016**, *7*, 875–884. [CrossRef]
39. Qu, T.; Zhang, Q.; Sun, S. Vehicle detection from high-resolution aerial images using spatial pyramid pooling-based deep convolutional neural networks. *Multimed. Tools Appl.* **2017**, *76*, 21651–21663. [CrossRef]
40. Tushar, A.K.; Ashiquzzaman, A.; Islam, M.R. Faster convergence and reduction of overfitting in numerical hand sign recognition using DCNN. In Proceedings of the 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC), Dhaka, Bangladesh, 21–23 December 2017; pp. 638–641.
41. Chollet, F. Keras. 2015. Available online: https://github.com/fchollet/keras (accessed on 19 August 2020).
42. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 448–456.
43. Ashiquzzaman, A.; Oh, S.; Lee, D.; Lee, J.; Kim, J. Compact Deeplearning Convolutional Neural Network based Hand Gesture Classifier Application for Smart Mobile Edge Computing. In Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Fukuoka, Japan, 19–21 Febuary 2020; pp. 119–123.