




Article

# A Bidirectional Arabic Sign Language Framework Using Deep Learning and Fuzzy Matching Score

Mogeeb A. A. Mosleh <sup>1,2</sup>, Adel Assiri <sup>3</sup>, Abdu H. Gumaiei <sup>4,\*</sup>, Bader Fahad Alkhamees <sup>5</sup> and Manal Al-Qahtani <sup>5</sup>

<sup>1</sup> Department of Software Engineering, Faculty of Engineering and Information Technology, Taiz University, Taiz 6803, Yemen; mogeebmosleh@taiz.edu.ye

<sup>2</sup> Department of Computer Science, Faculty of Computer Science and Information Technology, International University of Technology Twintech, Sana'a 7201, Yemen

<sup>3</sup> Department of Informatics for Business, College of Business, King Khalid University, Abha 61421, Saudi Arabia; adaseri@kku.edu.sa

<sup>4</sup> Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia

<sup>5</sup> Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia; balkhamees@ksu.edu.sa (B.F.A.); 443203302@student.ksu.edu.sa (M.A.-Q.)

\* Correspondence: a.gumaiei@psau.edu.sa

**Abstract:** Sign language is widely used to facilitate the communication process between deaf people and their surrounding environment. Sign language, like most other languages, is considered a complex language which cannot be mastered easily. Thus, technology can be used as an assistive tool to solve the difficulties and challenges that deaf people face during interactions with society. In this study, an automatic bidirectional translation framework for Arabic Sign Language (ArSL) is designed to assist both deaf and ordinary people to communicate and express themselves easily. Two main modules were intended to translate Arabic sign images into text by utilizing different transfer learning models and to translate the input text into Arabic sign images. A prototype was implemented based on the proposed framework by using several pre-trained convolutional neural network (CNN)-based deep learning models, including the DenseNet121, ResNet152, MobileNetV2, Xception, InceptionV3, NASNetLarge, VGG19, and VGG16 models. A fuzzy string matching score method, as a novel concept, was employed to translate the input text from ordinary people into appropriate sign language images. The dataset was constructed with specific criteria to obtain 7030 images for 14 classes captured from both deaf and ordinary people locally. The prototype was developed to conduct the experiments on the collected ArSL dataset using the utilized CNN deep learning models. The experimental results were evaluated using standard measurement metrics such as accuracy, precision, recall, and F1-score. The performance and efficiency of the ArSL prototype were assessed using a test set of an 80:20 splitting procedure, obtaining accuracy results from the highest to the lowest rates with average classification time in seconds for each utilized model, including (VGG16, 98.65%, 72.5), (MobileNetV2, 98.51%, 100.19), (VGG19, 98.22%, 77.16), (DenseNet121, 98.15%, 80.44), (Xception, 96.44%, 72.54), (NASNetLarge, 96.23%, 84.96), (InceptionV3, 94.31%, 76.98), and (ResNet152, 47.23%, 98.51). The fuzzy matching score is mathematically validated by computing the distance between the input and associative dictionary words. The study results showed the prototype's ability to successfully translate Arabic sign images into Arabic text and vice versa, with the highest accuracy. This study proves the ability to develop a robust and efficient real-time bidirectional ArSL translation system using deep learning models and the fuzzy string matching score method.

**Keywords:** bidirectional Arabic sign language; deep learning; convolutional neural network (CNN); fuzzy string matching score

**MSC:** 68T05; 03B52; 68T07; 68T09; 68T45; 68T50



**Citation:** Mosleh, M.A.A.; Assiri, A.; Gumaiei, A.H.; Alkhamees, B.F.; Al-Qahtani, M. A Bidirectional Arabic Sign Language Framework Using Deep Learning and Fuzzy Matching Score. *Mathematics* **2024**, *12*, 1155. <https://doi.org/10.3390/math12081155>

Academic Editor: Michael Voskoglou

Received: 1 March 2024

Revised: 7 April 2024

Accepted: 9 April 2024

Published: 11 April 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Deaf people suffer from several limitations that constrain them from performing their daily social activities due to the lack of proper communication with their society. A large population of deaf people lives without adequate support for the independent performance of their daily activities, especially in developing Arabic countries. According to recent World Health Organization (WHO) statistics, around 5% of people worldwide are considered deaf [1]. Statistics show that in the Arab region, about 17 million people are considered deaf, and most live in low- and middle-income countries [2]. Furthermore, a research study [3] reported that more than one million people with disabilities live in Saudi Arabia. Thus, people with hearing difficulties are considered a highly isolated population with difficulties interacting with their society to gain experience, knowledge, and relationships. Deaf and ordinary people mostly communicate using words, numbers, phrases, lip movements, and facial changes to express themselves. However, the lack of knowledge of sign language among ordinary people increases communication difficulties for deaf people, and it is impossible to force them to learn sign language for several reasons.

Recently, technology has attempted to bridge the gap by assisting deaf people to communicate and interact with their society. However, in the Arab region, the situation is relatively different due to several issues, such as the lack of government consideration for disabled people, the limited number of assistive technologies available, and the complexity of the Arabic language itself [4]. Researchers have tried to create a standard sign language for Arabic, consisting of 1600 basic and common Arabic signs [5]. However, the problem still exists because each country has modified sign language. Different signs can be found for the same word within distinct cities in the region [6]. A few types of research have been conducted on Arabic Sign Language (ArSL) translation, which suffers from several issues, such as the limitations of bidirectional translation, vocabulary words, and small datasets, due to the complexity of the Arabic language with the limitations of existing Arabic signs in context and an insufficient Arabic sign language dataset. Thus, this research is designed to facilitate communication between deaf and ordinary people by developing a robust framework for Arabic signs.

The significance and contributions of this study are to propose a framework for an automatic bidirectional ArSL translation system using various deep-learning methods with a fuzzy matching score model. Then, a prototype will be developed and evaluated based on the proposed framework. The objectives of this research can be listed as follows:

- Proposing a bidirectional Arabic sign translation framework using image processing approaches, CNN transfer learning, and the fuzzy string matching score method.
- Extracting the essential features of Arabic sign images using several effective transfer learning-based pre-trained CNN models.
- Designing a lightweight, efficient, and optimized Custom DNN model to classify Arabic sign language images into corresponding Arabic text words. The Custom DNN model is built with a minimum number of hidden layers and neurons.
- Developing a fuzzy string matching score-based method to translate Arabic words' synonyms and spelling errors into appropriate and meaningful Arabic sign language images.
- Constructing an ArSL dataset containing a large number of Arabic sign images of Arabic words that can be used for further research to enhance the ArSL recognition techniques.
- Evaluating the performance and efficiency of the ArSL prototype for each module individually.

The main contribution of this study is to propose a bidirectional framework for translating the Arabic sign images and input text into appropriate, meaningful words and sign images, respectively. The proposed framework is built to be suitable for real-time translation applications between deaf and ordinary people, and to be implemented on mobile devices. According to our previous research on sign language recognition [7,8], this study can be considered a preliminary study on adapting a bidirectional translator to facili-

tate bidirectional communication between the deaf and ordinary people. In addition, the study evaluates the performance and efficiency of the different CNN deep learning models within the context of sign language recognition. This study uses the fuzzy matching score model to translate the input text from ordinary people into sign images. It was adapted to improve the efficiency of sign language translation and solve some of the existing sign language issues.

This work is organized as follows: Section 2 briefly investigates the relevant previous studies on sign language. Section 3 gives the mathematical background and preliminaries about the methods used in the proposed framework. Then, the methodology of the proposed framework and prototype development is explained in Section 4. Section 5 presents the experimental results and discusses them. Finally, Section 6 provides the conclusion of the research work.

## 2. Literature Review

The deaf, hard-of-hearing, and hearing loss communities cannot express themselves well through speech or text, instead using sign language through hand gestures, facial expressions, and body movements. Sign languages are considered visual communication languages, which differ significantly from spoken languages in terms of both models and structure. Sign languages do not translate spoken words directly due to their constraints in sequences, structure, and syntax [9]. Sign language is considered a nonlinear language where communication is performed in parallel through several items such as hand direction and orientation, use of space, facial expression, eye, lip, and eyebrow movements, body posture, and facial expression [9].

Research has indicated that sign language differs from spoken language in terms of grammar, phonology, syntax, and morphology, where the phenotypics of hand signs have different morphemes. In addition, the morphological structure of sign language is not unique, making the linguistic features of the sign language variant significant [10].

Considerable research has recently been undertaken to develop a machine interpreter for recognizing different sign languages. Significant improvements in machine translation have been made with the assistance of technology [11,12]. There are several tools and techniques produced by technology to facilitate communication between deaf people and their communities. Sign Language Recognition (SLR) systems are classified into two main categories: glove-based and vision-based systems. Glove-based systems use hardware with electromechanical devices for gathering data input from deaf people's gestures. The glove is connected to several sensors to obtain the gesture data from the deaf person's hand and transfer the data into a computer device for recognition.

On the other hand, vision-based systems apply image processing with machine learning techniques to recognize sign language. Research has reported that a vision-based system is more convenient, flexible, and efficient than a glove-based system. Thus, most vision-based sign language translation systems are developed with three main modules: object detection, feature extraction, and sign recognition. Those three models are primarily implemented using image processing and machine learning classifiers, where image processing techniques are used for object detection and feature extraction.

In contrast, a machine learning algorithm identifies and recognizes signs. Some examples of these valuable techniques are R-CNN [13], histograms of oriented gradients (HOG) [14], and SSD techniques [15]. Some facial expression recognition models have also been used in addition to the hand pose to translate sign language.

A study performed by Nguyen and Ranganath on American sign language used the PCA model to extract the features of 28 facial expressions with a recursive tracker, where HMM and SVM machine learning was used to train and recognize the tracking results with improved recognition results for SLR [16]. Amrutha et al. [17] developed a mobile application to translate text from Indian Sign Language using the CNN model for feature extraction and classification from sign language images. This study proves the ability to apply CNN techniques in translating simple sign language. Rajam et al. [18] proposed a

model to recognize one of the South Indian sign languages using a set of 32 signs. They used an image-processing technique to extract the fingertip attributes and convert them into meaningful words. Four models used here as hand pattern recognition included data acquisition, signal detection, training, and a binary-to-text conversion. They used 320 images for training and 160 images for testing, and they reported the accuracy of their model as 98.125. This result was obtained due to the limited number of signs and images used, which was only 14.

Bhuyan et al. [19] adopted a vision-based method for compiling hand gestures to facilitate communication between ordinary and deaf people. A 12-megapixel webcam captures the hand movements in sequence to translate the hand-sign language into appropriate word meaning. Gandhi et al. [20] introduced a real-time Android application that converts hand signs into text and voice expressions. Background subtraction technology is used with the Blob detection algorithm to distinguish between pixel regions. They also used the Gaussian algorithm to improve the feature extraction process. They reported the ability of a proposed system to translate the sign into an equivalent text format. Lahoti et al. [21] proposed an Android application to convert American Sign Language into text using the YCbCr color model to break down pig skin and extract features from the input images. They used a support vector machine (SVM) for recognition, and their results showed an accuracy of 54%.

Recently, a deep learning technique was used in [22] using CNNs based on the Alex Net model for hand gesture recognition to enable communication with deaf or mute people, with high accuracy results. Aloysius and Geetha [23] tried to solve the problem of recognizing the isolated sign language (ISLR) with a solution extended to CSLR by shifting continuous signal sequences into text. They applied a traditional translation approach to identify the ongoing sign language (CSLR) using HMM, CRF, and DTW methods, improving sign recognition performance and efficiency. Imran et al. [24] proposed a framework for recognizing sign language using three types of motion blocks: DI, RGBMI, and MHI, excluding the required hand-splitting process. They achieved 70.7% accuracy when SVM was applied [24].

Arabic Sign Language (ArSL) receives inadequate attention from the research community due to several issues. One of the most critical issues is that Arabic is considered one of the most complex languages, with several grammatical rules, syntax, and linguistics constraints. An Automatic ArSL recognition system based on the glove model has been proposed by Assaleh et al. [25] with a polynomial networks classifier. They used real ArSL data collected from real deaf people and reported that their system achieved an accuracy rate of near 93%. El-Bendary et al. [26] proposed a vision-based model to translate sign language into text. Three main features were used to implement the hand position, and 50 orientation points with specific angles were extracted features. MLP neural network and minimum distance algorithms were used as classifiers to achieve 83.7% and 91.3% accuracy, respectively.

Samir et al. [27] performed a study to construct an ArSL database that included videos of many Arabic sign languages for translation purposes. They constructed a dictionary sign with 1216 numbers and alphabets, consisting of 531 sentences, and classified the Arabic sign into 27 words categorized and expressed using four different hand locations. Ahmed et al., 2016 [28] developed an automatic translation system that converts the Arabic sign language into an Arabic script (ATASAT). Their system relied on two Arabic sign language dictionaries to facilitate communication between deaf and ordinary people. They used three machine learning classifiers: KNN, MLP, and C4. Their system was helpful for educational purposes [28]. In 2017, they also produced an automatic translation system for converting Arabic sign language into written text using an annotated sign language corpus [29]. Their proposed system was designed using Arabic grammatical rules such as synonyms, derivatives, pluralism, masculinity, and coordination to construct a sentence by adding an annotation index. They reported that the proposed system demonstrated remarkable results in exploring finger dictation and solving some challenges found in



ArSL [29]. Recently, Luqman et al. [30] introduced a machine translation system for Arabic sign language that allowed manual interpreters to preserve the structure and meaning of Arabic sentences. Their system significantly reduced long sentences without losing the essential words to convey the meaning.

Furthermore, Aly et al., 2020 [31] introduced a framework for ArSL recognition using three deep learning networks with adaptive instant scaling algorithms for real-time processes to recognize all hand-sign gestures. Their framework suggested a combination model of semantic segmentation networks with SOM and two-way deep networks. They showed an increase in translation performance of 70% and an average accuracy of nearly 89.5% [31]. Furthermore, Kamruzzaman et al. [32] published a study that translated ArSL into Arabic voice using CNN to analyse and recognize the input images. They reported that their system achieved great accuracy of approximately 90%. Recently, research has tried to come up with gaps and limitations of Arabic sign language such as dataset size and availability, low efficiency, computation complexity, one-directional translation, facial expression, and different meanings for signs in each country. Balaha et al., 2022 proposed a vision-based deep learning approach to translate 20 words of Arabic sign language by employing both the CNN and RNN models. They customized several existing datasets to create their own dataset with 20 Arabic signs for model evaluation. They reported that the proposed model achieved an accuracy rate of 93.4% [33]. Nahar et al., 2023 utilized 12 models of deep learning to translate Arabic sign images into voice. They used 10 sign images containing the signs for Arabic numbers to create the dataset. Their experiment showed that the best accuracy result was 93.6% obtained by the VGG16 model [34]. AbdElghfar et al., 2023 proposed a model to recognize Arabic sign language letters for the learning purposes of the deaf and dumb. The dataset included 14 letter signs with 24,137 sign images. They reported that the proposed model achieved better performance than existing models [35]. Amor A. et al., 2023 introduced electromyography (EMG) as hand gesture recognition to construct the Arabic sign dataset for letters and numbers only. They reported the promise of using such devices for real-time sign translation systems. However, no experiment was conducted to evaluate their dataset for real-time sign language application purposes [36]. Alsulaiman [37] et al., 2024 performed a survey using 17 Arabic sign language datasets to develop a suitable framework for building an Arabic sign language dataset. They created a large dataset for Saudi sign language including 145,035 samples for 293 signs with 10 different domains. They proposed a CGCN model architecture for sign language recognition for use in building their own dataset. In 2024, El Kharoua and Jiang introduced a CNN model for the recognition of Arabic alphabet sign language using the AASL dataset. They used several methods for data preparation, including data cleaning, resizing, background removal, augmentation, zooming, and flipping. They reported that their methods achieved outstanding accuracy, reaching 97.4% [38].

### 3. Preliminaries

This work uses deep neural networks (DNNs) and a transfer learning technique to recognize the meaning of sign language images captured by deaf people and return their corresponding Arabic words to ordinary people. Furthermore, we use the fuzzy string matching score to search for Arabic words in the database and return the stored sign language images to deaf people. DNNs are frequently used in the research field of automatic classification tasks due to their adaptive nature and accuracy [39]. State-of-the-art architectures and pre-trained networks can be applied to extract the essential features of sign language images and build an effective classifier to classify them. Fuzzy text/string matching is a technique for identifying strings that match a pattern approximately (rather than perfectly). In other words, fuzzy string matching is a form of search that finds matches even when users misspell or mispronounce words or enter/pronounce only partial phrases in the search. Also, the fuzzy string matching technique is beneficial when one wishes to obtain the degree of association of a word  $x$  with its index term word  $x_k$  to accurately return the sign language image  $y$ ; that is, the degree of the meaning for the index

term word  $x_k$  corresponds to the meaning of the provided term  $x$ . The purpose of this connection is to address the issue of synonyms among index words. This relation facilitates the identification of relevant terms for a particular query that would not otherwise be identified. It occurs if a word characterized by a synonymous index term is determined by an index term identical to an index term found in the query.

The following subsections introduce initiations and mathematical definitions of deep neural networks, fuzzy sets, and the concept of fuzzy string matching.

### 3.1. Deep Neural Networks (DNNs)

McCulloch and Pitts [40] had the idea to bring artificial intelligence into the world in 1943. At that time, they aimed to create an algorithmic method of learning that mimicked the functioning of the human brain. Because the brain consists of neurons with countless connections, they introduced so-called artificial neurons as building blocks [41]. In its most basic form, a neuron in the human brain is made up of dendrites that send signals to the SOMA while being amplified/scaled by the structural features of the corresponding dendrites. In the neuron’s SOMA, the incoming impulses are collected, and a choice is made of whether or not to fire to other neurons and, if so, with what intensity [41].

In this section, we give a mathematical definition of an artificial neuron with its weight and bias parameter values, which need to be trained.

**Definition 1.** An artificial neuron network with input  $x_1, \dots, x_n$ , activation function  $\rho : R \rightarrow R$ , bias  $b \in R$ , and weights  $w_1, \dots, w_n \in R$ , is defined as a function  $g : R^n \rightarrow R$ , given by [42]:

$$g(x_1, \dots, x_n) = \rho \left( \sum_{i=1}^n x_i w_i + b \right) = \rho(\langle x, w \rangle + b) \tag{1}$$

Some examples of activation functions with their mathematical equations are given below [42]:

- Rectifiable Linear Unit (ReLU):

$$\rho(x) = \max(x), \tag{2}$$

- Step function:

$$\rho(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0. \end{cases} \tag{3}$$

- Sigmoid function:

$$\rho(x) = \frac{1}{1 + e^{-x}}. \tag{4}$$

The most straightforward fundamental activation function is the step function, which results in a no/yes decision. The sigmoid function offers a smoother option. However, the ReLU is by far the most commonly utilized activation function in almost all applications due to its simple piecewise linear construction, which is useful in the training procedure while still allowing for outstanding performance [42].

A feedforward artificial neural network is created by concatenating artificial neurons with affine linear maps and activation function compositions. This yields the following definition:

**Definition 2.** Suppose  $d \in N$  is the dimension of the input layer,  $L$  the number of layers,  $N_0 = d$ ,  $N_l, l = 1, \dots, L$  is the dimensions of the hidden and last layers, and  $\rho : R \rightarrow R$  is a

non-linear activation function. For  $l = 1, \dots, L$ , the  $T_l$  will be the affine-linear functions, given as follows [42]:

$$T_l : R^{N_{l-1}} \rightarrow R^{N_l}, \quad T_l x = w^{(l)}x + b^{(l)}, \tag{5}$$

with  $w^{(l)} \in R^{N_l \times N_{l-1}}$  being the weight matrices and  $b^{(l)} \in R^{N_l}$  is the bias vector of the  $l$ th layer. Also,  $\Phi : R^d \rightarrow R^{N_L}$  can be written as [42]:

$$\Phi(x) = T_L \rho(T_{L-1} \rho(\dots \rho(T_1(x)) \dots)), \quad x \in R^d \tag{6}$$

By means of Equation (6), we define what are called DNN.

A deep convolutional neural network (CNN) is one type of DNN, widely used in various image-related tasks such as object detection, image classification, and image segmentation. Let us introduce a simple and abstract description of the CNN structure for classification, running layer by layer in a forward pass with padding  $p$  and stride  $s$  as follows [43]:

$$x^1 \xrightarrow[f^1]{Conv} x^2 \xrightarrow[f^2]{Conv} x^3 \xrightarrow{Pooling} \dots \rightarrow x^l \xrightarrow{Flatten} x^{l+1} \xrightarrow[w^{l+1}]{Dense} \dots \rightarrow x^L \xrightarrow{Softmax} z$$

Suppose  $x^{l-1}$  denotes the input feature map (image),  $f$  is the filter (also called weight or kernel),  $b$  is the bias, and  $x^l$  denotes the output feature map. The convolution operation is defined as [43]:

$$x^l = \rho(f^l * x^{l-1} + b^l), \tag{7}$$

where  $*$  denotes the convolution operation,  $+$  denotes an element-wise addition operation after convolution, and  $\rho$  represents an activation function. It transforms each coefficient  $\alpha$  of the matrix  $f^l * x^{l-1}$ , and can be applied element-wise to satisfy the non-linearity of the input. A usual choice is a rectifier such as the ReLU function given in Equation (2) to be  $\rho(\alpha) = \max(\alpha, 0)$  for  $\alpha \in R$ , but it can also be a sigmoid or a modulus  $\rho(\alpha) = |\alpha|$  where  $\alpha$  may be complex [44].

Suppose a 3D tensor input  $x^{l-1}$  has height  $m^{l-1}$ , width  $n^{l-1}$ , and the number of channels  $nc^{l-1}$ . Also, a 3D tensor filter  $f^l$  has height  $q^l$ , width  $r^l$ , and the number of filters  $nf^l$ . The dimension of the 3D tensor output  $x^l$  can be given as:

$$x^l_{(m_t^l, n_t^l, nf^l)} = \rho \left( f^l_{(q^l, r^l, nc^l)} * x^{l-1}_{(m^{l-1}, n^{l-1}, nc^{l-1})} + b^l_{(m_t^l, n_t^l, nf^l)} \right) \tag{8}$$

The  $m_t^l$  and  $n_t^l$  are the new height and width of the 3D tensor feature map output  $x^l$ , computed using the following equations:

$$m_t^l = \frac{m^l + 2p - q^l}{s} + 1 \tag{9}$$

$$n_t^l = \frac{n^l + 2p - r^l}{s} + 1 \tag{10}$$

The convolution operation term in Equation (8) can be represented as a 3D tensor vector  $v$  in Equation (11) and can be rewritten as in Equation (12):

$$v^l_{(m_t^l, n_t^l, nf^l)} = f^l_{(q^l, r^l, nc^l)} * x^{l-1}_{(m^{l-1}, n^{l-1}, nc^{l-1})} \tag{11}$$

$$x^l_{(m_t^l, n_t^l, nf^l)} = \rho \left( v^l_{(m_t^l, n_t^l, nf^l)} + b^l_{(m_t^l, n_t^l, nf^l)} \right) \tag{12}$$

Because  $f^l$  is linear,  $v^l_{(m^l, n^l, n^{f^l})}$  may be written as a sum of convolutions:

$$v^l_{(i=1\dots m^l, j=1\dots n^l, d=1\dots n^{f^l})} = \sum_{i=1}^{m^{l-1}} \sum_{j=1}^{n^{l-1}} \sum_{d=1}^{n^{f^l-1}} f^l(i, j, d) \times x^{l-1}(m^{l-1} - i, n^{l-1} - j, d) \tag{13}$$

Pooling layers are often used to reduce or down-sample the feature maps' dimensions, decreasing the computation's complexity and only extracting the dominant features. Two types of pooling operators are widely used. Average pooling is one of the pooling operators that take the average value of all features in the pooling window. Max pooling is another commonly used pooling operator. Suppose that the height and width of the window size for the pooling layer are  $pH$  and  $pW$ . The average and max pooling for each channel of the input feature map  $x^{l-1}$  are given in Equations (14) and (15), in which the number of channels of the output feature map  $x^l$  will have the same number of channels for the input feature map  $x^{l-1}$ , and its height and width will be  $m^l = m^{l-1} - pH + 1$  and  $n^l = n^{l-1} - pW + 1$ .

$$x^l_{(i=1\dots m^l, j=1\dots n^l)} = \frac{1}{pH \times pW} \sum_i^{m^l} \sum_j^{n^l} x^{l-1}(i + pH, j + pW) \tag{14}$$

$$x^l_{(i=1\dots m^l, j=1\dots n^l)} = \max_{0 \leq i \leq m^l, 0 \leq j \leq n^l} x^{l-1}(i + pH, j + pW) \tag{15}$$

After several convolutional and pooling layers, the extracted features  $x^l$  are flattened into a vector with size  $(1, a)$ , where  $a = m^l \times n^l \times n^{f^l}$ , and passed through one or more fully connected layers (dense layers) with several hidden neurons  $h$ , weight  $w^{l+1}$ , and bias  $b^{l+1}$ . The output vector  $x^{l+1}$  is computed as follows:

$$x^{l+1}_{(1, h)} = \rho \left( x^l_{(1, a)} \times w^{l+1}_{(a, h)} + b^{l+1}_{(1, h)} \right) \tag{16}$$

The multiplication operation  $x^l_{(1, a)} \times w^{l+1}_{(c, h)}$  can be represented as a vector  $u^{l+1}$  with size  $(1, h)$  in Equation (17) and computed using Equation (1) as rewritten in Equation (18):

$$x^{l+1}_{(1, h)} = \rho \left( u^{l+1}_{(1, h)} + b^{l+1}_{(1, h)} \right) \tag{17}$$

$$u^{l+1}_{(1, h)} = \sum_{j=1}^a x^l_{(1, j)} \times w^{l+1}_{(j, h)} \tag{18}$$

The last layer  $z$  is a loss layer, which has a cost or loss function used to measure the discrepancy between the prediction of  $x^L$  and the corresponding target (ground-truth) value  $t$  for the input  $x^1$ .

Suppose the classification task has  $c$  labels/classes. A frequently used approach is to output  $z$  as a  $c$  dimensional vector that encodes the prediction (the posterior probability of  $x^1$  derives from the  $i$ th class) [43]. To make  $z$  a probability mass function, the  $(L)$ th processing will be as a transformation of the softmax function of  $x^L$ , given as:

$$z_{(1, j)} = \frac{e^{x^L_{(1, j)}}}{\sum_{j=1}^c e^{x^L_{(1, j)}}} \tag{19}$$

$$z_{(1, j)} = \operatorname{argmax}_{0 \leq j \leq c} x^L_{(1, j)}, \tag{20}$$

where  $c$  is the number of class labels and  $z_{(1, j)}$  is the output of the  $j$ th neuron in the softmax layer.

These equations define the core functions of a deep CNN model. The parameters (weights, filters, and biases) are trained using a back-propagation method, which may be

generated using stochastic gradient descent and regularization techniques like dropout throughout the training phase. The training method minimizes a predetermined loss function (such as cross-entropy loss) between expected outputs and ground truth labels. As in several other learning methods, the CNN’s parameters are optimized to minimize the loss  $z$ , i.e., the CNN model’s prediction aims to match the ground-truth labels. Assume one training instance  $x^1$  is provided for training such parameters. The CNN network will be run in both directions during the training process. In the forward pass, the network runs to achieve a prediction and obtain  $x^L$  using the current CNN parameters [43]. Then, instead of outputting the predicted class, it compares the predicted class with the target class  $t$  corresponding to  $x^1$ . This means continuing the forward pass until the last loss layer. Finally, a loss of  $z$  will be achieved. The loss  $z$  is a supervision indicator to guide the model’s parameters to be updated [43]. The SGD method used for modifying the parameters is defined as [43]:

$$w^i \leftarrow w^i - \eta \frac{\partial z}{\partial w^i} \tag{21}$$

The learning procedure of CNN can depend on the vector calculus and chain rule. Suppose  $z$  is a scalar (i.e.,  $z \in R$ ) and  $y \in R^H$  is a vector. So, if  $z$  is a function of  $y$ , then the partial derivative of  $z$  concerning  $y$  is a vector, defined as [45]:

$$\left(\frac{\partial z}{\partial y}\right)_i = \frac{\partial z}{\partial y_i} \tag{22}$$

Explicitly,  $\left(\frac{\partial z}{\partial y}\right)$  is a vector with the same size as  $y$ , and its  $i$ th element is  $\left(\frac{\partial z}{\partial y}\right)_i$ . Also, note that  $\left(\frac{\partial z}{\partial y^T}\right) = \left(\frac{\partial z}{\partial y}\right)^T$ . Furthermore, presume  $x \in R^W$  is another vector, and  $y$  is a function of  $x$ . Then, the partial derivative of  $y$  concerning  $x$  is defined as [45]:

$$\left(\frac{\partial z}{\partial x^T}\right) = \left(\frac{\partial z}{\partial y^T}\right) \left(\frac{\partial y}{\partial x^T}\right) \tag{23}$$

Some network topologies have shown instabilities [46], where the addition of tiny perturbations on  $x$  can result in huge fluctuations of  $x_i$ . This occurs when the norms of the matrices  $w_i$  are greater than one and hence magnified when cascaded. However, transfer learning [47] demonstrates that deep networks have a high level of stability. If the final classification layers are trained on a new (domain-specific) dataset, a deep network layer  $x_i$  that has been tuned on specific training datasets might approximate distinct classification functions. This suggests that it has acquired stable structures that can be applied to comparable learning tasks.

### 3.2. Fuzzy Set and Fuzzy String Matching Score

In fuzzy set theory, a set can be defined by its ambiguous boundaries, which means that uncertainty exists about the location of these boundaries. A classical set theory allows the membership of the elements in a set of binary terms  $\{0, 1\}$ . A fuzzy set theory is an extension of the classical set theory, in which the membership of an element is a function valued in the interval  $[0, 1]$  [48].

Suppose we use the crisp and fuzzy sets’ membership function (discrimination function or characteristic function). In this case, we can characterize whether an element  $x$  is in a set  $A$  or not using the following definitions:

**Definition 3** (Membership function of crisp set). *For a set  $A$ , we define a membership function  $\mathcal{M}_A$  such as [48]:*

$$\mathcal{M}_A = \begin{cases} 1 & \text{if and only if } x \in A \\ 0 & \text{if and only if } x \notin A \end{cases} \tag{24}$$



We can say that the function  $\mathcal{M}_A$  in the crisp set maps the elements in the universal set  $X$  to the set  $\{0, 1\}$ .

$$\mathcal{M}_A : X \rightarrow \{0, 1\} \tag{25}$$

**Definition 4** (Membership function of fuzzy set). In fuzzy crisp sets, each element is mapped to  $[0, 1]$  by a membership function, represented as [48]:

$$\mathcal{M}_A : X \rightarrow [0, 1], \tag{26}$$

where  $[0, 1]$  is the interval of real numbers between 0 and 1 (including 0, 1).

Consequently, the fuzzy set is a ‘vague boundary set’ compared with a crisp set. Fuzzy string/text matching is a method used to find strings that match a pattern approximately (rather than exactly) [49]. In other words, fuzzy string matching is a search that tries to find matches even when users misspell/mispronounce words or enter/pronounce only partial words for the search. Several fuzzy string matching algorithms include Levenshtein distance, Jaccard similarity, Jaro–Winkler, and Longest Common Subsequence (LCS) [49]. This work selects a Levenshtein distance algorithm due to its versatility and simplicity, allowing for seamless integration into several applications and programming languages.

Moreover, it is robust for comparing textual data, making it valuable for various data analytics and artificial intelligence (AI) tasks. The Levenshtein distance [50] is a string measure for obtaining the difference between two words. One of the most effective fuzzy string-matching metrics is the Levenshtein distance. The Levenshtein distance between two words is the minimum number of single-character edits (deletions, insertions, or substitutions) for changing one word into another [51]. This subsection presents a mathematical description of the Levenshtein distance algorithm.

**Definition 5** (Levenshtein Distance). The Levenshtein distance measure between two words  $S_1$  and  $S_2$  of length  $|S_1|$  and  $|S_2|$ , correspondingly, is given by [51]:

$$\text{lev}(S_1, S_2) = \begin{cases} |S_1|, & \text{if } |S_2| = 0, \\ |S_2|, & \text{if } |S_1| = 0, \\ \text{lev}(\text{tail}(S_1), \text{tail}(S_2)) & \text{if } \text{head}(S_1) = \text{head}(S_2), \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(S_1), S_2) \\ \text{lev}(S_1, \text{tail}(S_2)) \\ \text{lev}(\text{tail}(S_1), \text{tail}(S_2)) \end{cases} & \text{otherwise} \end{cases}, \tag{27}$$

where the tail of a string word  $S$  is a string of all but the first character of word  $S$ , and  $\text{head}(S)$  is the first character of word  $S$ . Either the notation  $S[n]$  or  $S_n$  is used to refer to the  $n$ th character of the string word  $S$ , counting from 0; thus,  $\text{head}(S) = S = S[0]$ .

The first element in the minimum matches the deletion (from  $S_1$  to  $S_2$ ), followed by insertion and replacement. This definition is the same as the naïve recursive implementation. The process computes the Levenshtein distance in a matrix row-by-row, in which each cell represents the edit distance between a prefix of  $S_1$  and a prefix of  $S_2$ . Precisely, the function  $\text{lev}(i, j)$  takes the value of one if the  $i$ th word of  $S_1$  is different from the  $j$ th word of  $S_2$ , and zero otherwise. The final edit distance between  $S_1$  and  $S_2$  is given by the value in the bottom-right corner  $\text{lev}(S_1, S_2)$  of the Levenshtein distance matrix [51].

#### 4. Proposed Framework

In this study, an Arabic sign language translation framework is mainly designed using image processing, deep learning, and fuzzy logic methods. The main aim of this study is to propose a bidirectional framework for developing an automatic translation prototype of Arabic sign language into corresponding meaningful words and vice versa. The proposed

framework is designed to include two main modules for automatic bidirectional translation, as shown in Figure 1 below.

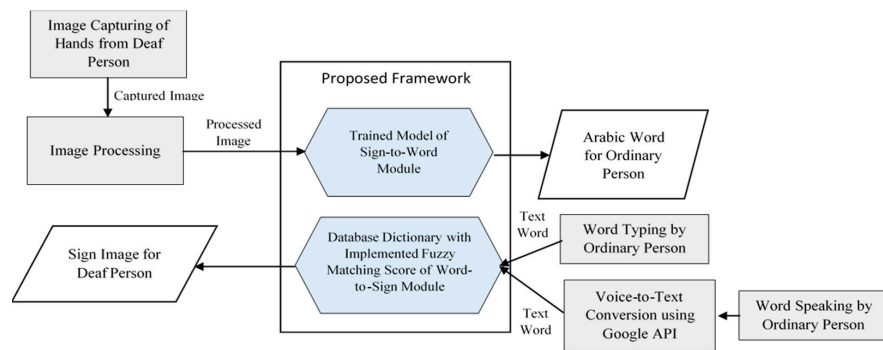


Figure 1. Proposed framework for bidirectional Arabic sign language.

The framework for translating the sign language bi-directionally is proposed. The first module of the framework is developed to translate the sign image of deaf people into text understood by ordinary people. The second module of the framework is built to translate the input Arabic text for ordinary people into sign images to be understood by deaf people. The two main modules will be briefly discussed in the following sections.

#### 4.1. Sign-to-Word Module

This module is developed to support the translation process between deaf and ordinary people. It consists of dataset collection and an image processing step, CNN implementations and a feature extraction model, a training and testing model, and system developments, as shown in Figure 2 below.

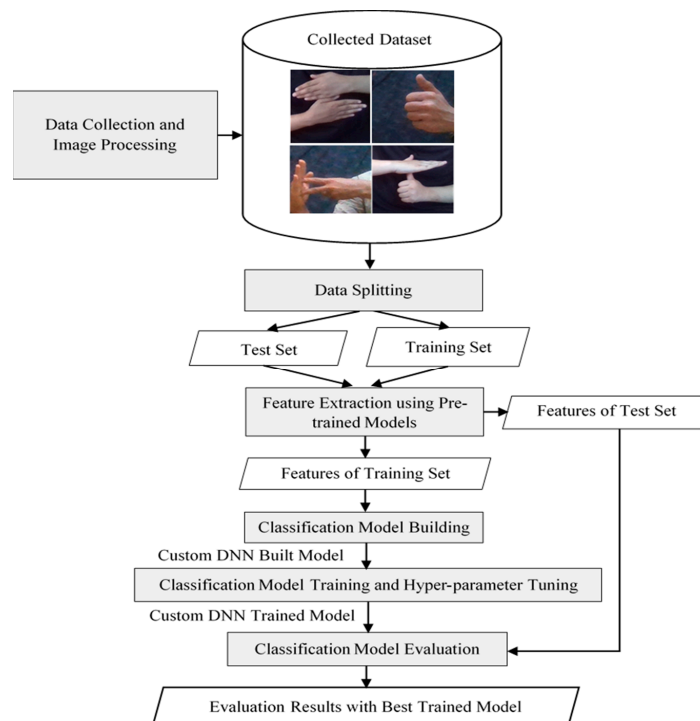


Figure 2. The sign-to-word translation module.

##### 4.1.1. Dataset Collection and Image Processing

One of the objectives of this study is to create a new dataset for Arabic sign language (ArSL) that can be used as a base for future state-of-the-art studies. The criteria used to develop the ArSL dataset was performed using the hand gestures from “The Unified Arabic

Dictionary". The hand-sign images are captured in unconstrained environments to create the dataset with the following criteria:

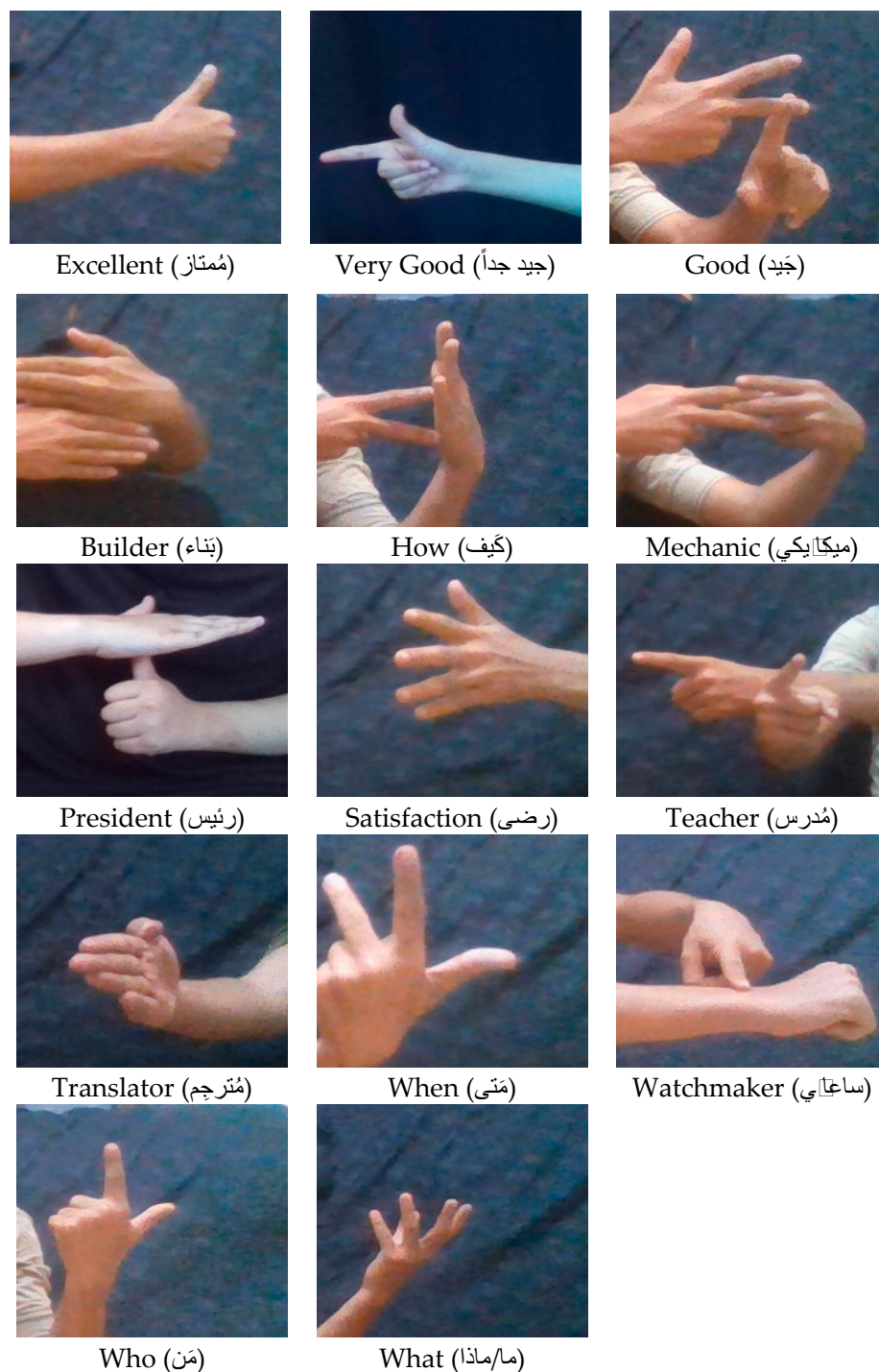
- Hand signs taken represented different words only without any facial expression.
- Deaf and ordinary people captured the dataset's hand-sign images to represent bidirectional ArSL translation.
- The dataset images included different hand gestures, moves, and positions.
- The dataset images were captured by using the front camera of a Samsung Android system to represent real-time translation between deaf and ordinary people.
- The dataset hand images were captured from different distances, angles, backgrounds, and light sources.
- The dataset's hand images were taken with good image quality and hand-sign focus.

Image acquisition was performed using the front camera of a Samsung Note 9, where the captured image resolution is  $2640 \times 1980$  pixels. Two image processing methods were applied mainly to fix the dataset criteria issues, including image enhancement and segmentation. The image enhancement method improves the intensity and quality of captured images. These images are acquired with different brightness and contrast values due to the conditions of indoor and outdoor environments, such as changes in lights, shadows, and background. It uses an adaptive histogram equalization (AHE) algorithm to enhance the dataset images by automatically computing images' histograms and correcting their contrast and brightness. Because of camera shake and resolution changes, some images might be noisy and blurry; hence, the AHE effectively deals with these issues. An image segmentation model also extracts hand gestures from different backgrounds. A slope difference distribution (SDD) method has been implemented in this study and has recently been considered the most accurate segmentation technique for hand gestures [52].

Then, the extracted hand-sign images were rotated automatically using the angle of inclination with the object auto alignment function into the appropriate hand gesture position [53]. The final hand-sign images were resized into  $128 \times 128$  resolution to increase the performance of subsequent feature extraction and classification steps. The processed sign images were organized and labeled to create a preliminary Arabic sign language (ArSL) dataset for training and testing purposes. The dataset contained 7030 sign images with 14 labels, which are Arabic words, as shown in Table 1. The dataset creation process was tedious and time-consuming due to the study criteria constraints, such as removing duplicate images, corrupted images, and images with unwanted shadows, where many images were excluded during the selection process. Figure 3 shows examples of dataset sign image results from the previous steps.

**Table 1.** Final dataset of selected sign images.

Class Name	Class Label	Number of Instances
builder	0	341
excellent	1	442
good	2	419
how	3	427
mechanic	4	337
president	5	640
satisfaction	6	444
teacher	7	662
translator	8	469
very good	9	594
watch maker	10	634
what	11	509
when	12	534
who	13	578
Total		7030



**Figure 3.** Examples of hand-sign images used in this study.

#### 4.1.2. Data Splitting

This step used two splitting procedures to build and evaluate the developed models. In each procedure, the dataset was divided randomly into three sets using a holdout technique. The first splitting procedure divided the dataset into 80% for training and 20% for testing. Then, from the 80% of the training set, 20% was taken randomly as a validation set, as illustrated in Table 2. The dataset-splitting process of this procedure was performed for models' training and evaluation of the proposed sign-to-word module.

**Table 2.** The distribution of classes in the training, validation, and test sets using the first splitting procedure (80:20).

Arabic Class Name	English Class Name	Class Label	Training Set	Validation Set	Test Set
بَناء	builder	0	224	54	63
مُمتاز	excellent	1	269	76	97
جيد	good	2	264	68	87
كَيْف	how	3	278	72	77
ميكانيكي	mechanic	4	209	55	73
رئيس	president	5	401	100	139
رضى	satisfaction	6	280	66	98
مُدّرس	teacher	7	420	117	125
مُترجم	translator	8	315	59	95
جيد جداً	very good	9	392	92	110
ساعاتي	watchmaker	10	404	109	121
ماذا/ما	what	11	338	68	103
متى	when	12	349	92	93
من	who	13	356	97	125
Total			4499	1125	1406

For generalizing the capabilities of the developed models and confirming the performance results of the first splitting procedure, a second splitting procedure was applied to divide the dataset into 70% for training and 30% for testing. From the 70% training set, 10% was taken randomly as a validation set, as illustrated in Table 3.

**Table 3.** The distribution of classes in the training, validation, and test sets using the second splitting procedure (70:30).

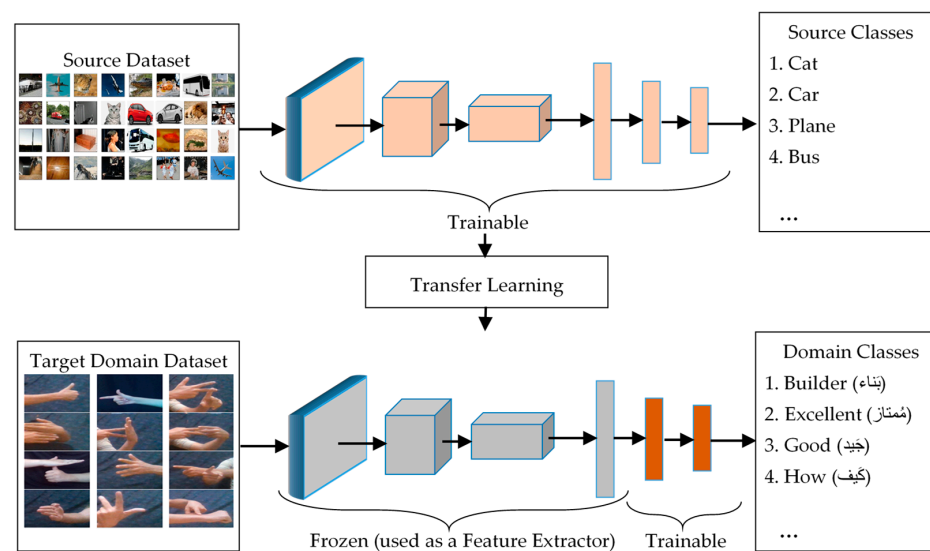
Arabic Class Name	English Class Name	Class Label	Training Set	Validation Set	Test Set
بَناء	builder	0	214	22	105
مُمتاز	excellent	1	284	26	132
جيد	good	2	252	41	126
كَيْف	how	3	278	33	116
ميكانيكي	mechanic	4	214	19	104
رئيس	president	5	394	49	197
رضى	satisfaction	6	275	28	141
مُدّرس	teacher	7	422	44	196
مُترجم	translator	8	299	27	143
جيد جداً	very good	9	383	45	166
ساعاتي	watchmaker	10	397	38	199
ماذا/ما	what	11	319	39	151
متى	when	12	338	46	150
من	who	13	359	36	183
Total			4428	493	2109



#### 4.1.3. Feature Extraction Using Pre-Trained Models

In this subsection, we extracted the crucial features from sign images using transfer learning-based pre-trained CNN models. Feature extraction is an important step, especially for small datasets, which is considered a challenging task. Hence, the transfer learning techniques made it easier than training the CNN model from scratch.

Transfer learning employs previously learned knowledge from the source domains to obtain high performance in diverse but related target domains [54,55]. In transfer learning, a domain is formally described as  $D = \{F, P(F)\}$ , where  $F$  is a feature space and  $P(F)$  is a marginal distribution for  $F = [f_1, f_2, \dots, f_n] \in \mathbb{R}^m \times n$ . A task in a domain  $D$  can be written formally as  $T = \{y, g(\cdot)\}$ , where  $y$  is a label space and  $g(\cdot)$  denotes a decision boundary function [54]. Transfer learning attempts to enhance the learning of the decision boundary function  $f(\cdot)$  in a target domain (TD) and a target learning task (TLT), utilizing information from a source domain (SD) and a source learning task (SLT), where  $SD \neq TD$  or  $SLT \neq TLT$ . As illustrated in Figure 4, we assume that ImageNet and Arabic Sign Language are the source and target domain datasets, respectively.



**Figure 4.** The context of our feature extraction method using the transfer learning concept.

Based on the concept of transfer learning and the mathematical fundamentals of deep neural networks given in Section 3, the pre-trained CNN transfer learning models have already been trained and tested on the source domain dataset through a source learning task. Transferring the learned parameters of pre-trained CNN models as feature extractors to extract the essential features of our target domain dataset reduces the training costs. It improves TLT in the Arabic Sign Language dataset. The final extracted bottleneck feature matrix size from pre-trained CNN feature extractors depends on the filter size. It is given by multiplying the filter height, width, and the number of channels. The turn filter size is based on the size of the input image and the number of channels subjected to pre-trained CNN feature extractors used. For example, if the image size is  $128 \times 128$ , as in our case, and the VGG16 is used for feature extraction, the dimension of the extracted bottleneck feature matrix is  $4 \times 4 \times 512$ .

In state-of-the-art transfer learning, there are several pre-trained CNN models in which the kernels are defined by different sizes for object classification from the images. Such pre-trained CNN models can extract imperative features from images for various applications. They can extract domain-specific features for task transfer learning, as Pan and Yang proposed [54].

In our study, pre-trained CNN models were adopted to extract the image features of hand Arabic signs to translate the input hand-sign images into text words based on hand-sign attributes. The pre-trained CNN models extracted the significant features from

the Arabic sign images, capturing their properties and characteristics. There are three main types of features: low-level features, which are the essential elements of hand-sign images, such as edges, pixels, contours, or corners extracted by the lower layers of pre-trained CNN models; intermediate features, such as textures and patterns of hand-sign images extracted by intermediate layers of pre-trained CNN models; and high-level features like more complex and abstract features extracted by higher layers of pre-trained CNN models, allowing the proposed Custom DNN model to recognize hands shape and fingers at different levels of abstraction. The utility of those features could improve the accuracy of the proposed model. Among the existing pre-trained CNN models, five famous architectures were selected for their efficiency in performing model training from scratch. These pre-trained models are described as follows:

1. DenseNet121 Model

DenseNet121 is a CNN-based model architecture introduced in 2016 by Huang et al. [56] to achieve high performance in image classification tasks. It provides accessible communication between layers within the condensed block, facilitating extensive reusing between layers by allowing each layer to access feature maps with the previous layer. Research has reported that the DenseNet121 model is suitable because it is less prone to over-fitting and adaptability for a broad type of domain [57]. It has recently been used widely in many transfer learning-based applications. DenseNets models are developed with bottleneck blocks to include fewer parameters than the other traditional models without the need to learn redundant feature maps. The dense block of DenseNets solves some training issues by allowing each layer to access the gradients directly from the loss function, and the input image. It introduces dense connections between layers, where each layer receives the feature maps from all preceding layers. The feature maps are concatenated along the channel dimension. The mathematical expression for the dense block in DenseNet is given as:

$$H[l] = \text{Concatenate}(H[0], H[1], \dots, H[l-1]), \quad (28)$$

where  $H[l]$  denotes the output feature maps of the  $l$ th layer.

2. ResNet152 Model

ResNet152 is a CNN-based model introduced in 2015 and designed with 152 hidden layers. This model is efficient for image feature extraction with more robotic accuracy results. The main issue with this model is its size and time limitations; however, it is considered adequate for training deep networks with less complex computation. We trained this model using a set of layers: Sequential, Conv2d, BatchNorm2d, Max-pool2d, and Bottleneck, with Adam optimizer and softmax activation function [58]. ResNet introduces residual connections that allow the model to learn residual mappings, making it easier to optimize deep networks. The residual block in ResNet can be expressed mathematically as:

$$H[l+1] = H[l] + F(H[l]), \quad (29)$$

where  $H[l]$  represents the input to the  $l$ th layer,  $H[l+1]$  is the output of the  $l$ th layer, and  $F(H[l])$  denotes the residual mapping.

3. MobileNetV2 Model

MobileNetV2 is a computationally efficient CNN, a more robust and efficient convolution neural network, designed to work on mobile devices. MobileNetV2 is upgraded from the MobileNetV1 model with a slight difference in structure, which contributes to its effectiveness by reducing complexity cost and network model size. The structure layer of this model consists of 32 filter layers followed by 19 residual bottleneck layers. The first layer is called the expansion layer, which has a  $1 \times 1$  convolution layer that widens the channels; the second layer is a deep convolution layer and input filtering; and the third layer, called the drop layer, is a  $1 \times 1$  raster convolution [59]. MobileNet is designed for efficiency and low computational cost by employing depth-

wise separable convolutions. The mathematical expression for a depth-wise separable convolution in MobileNet is given as:

$$DWConv = \text{DepthwiseConv}(\text{Conv}(x)), \quad (30)$$

where  $\text{Conv}(x)$  represents a standard convolution operation on input  $x$ , and  $\text{DepthwiseConv}$  represents depth-wise convolution.

#### 4. Xception Model

It is a deep convolutional neural network developed by Google with a depth-wise separable convolution model and shortcuts between convolution blocks. It's an interpretation of Inception modules to be used as an intermediate step in regular convolution and the depth-wise separable convolution operation. The depth-wise separable convolution is considered an Inception module with many towers. This algorithm uses depth-wise separable convolutions to replace the Inception model and develop a novel deep convolutional neural network. It is perceived to be much more efficient in computation time [60]. XceptionNet is an extension of the Inception module that replaces the standard convolutional layers with depth-wise separable convolutions. The mathematical expression for a depth-wise separable convolution operation in XceptionNet is given as:

$$DWConv = \text{DepthwiseConv}(\text{Conv}(x)), \quad (31)$$

where  $\text{Conv}(x)$  represents a standard convolution operation on input  $x$  and  $\text{DepthwiseConv}$  represents depth-wise convolution.

#### 5. Inception-v3 Model

It is a convolutional neural network architecture in the Inception family built with several improvements on existing Inception CNN models. The improvement of the model concerns label smoothing, factorized  $7 \times 7$  convolutions, and an auxiliary classifier to propagate label information. Inception-v3 is a CNN model built for image recognition that achieves greater than 78.1% accuracy on the ImageNet dataset. Multiple researchers have developed it over the last few years. It includes symmetric and asymmetric building blocks with convolutions, max pooling, average pooling, concatenations, dropouts, and fully connected layers. It uses batch normalization for activation inputs, where softmax is used to compute loss [61]. It combines the Inception module with residual connections, leveraging the benefits of both architectures. The mathematical expression for an Inception-residual block is written as:

$$H[l + 1] = H[l] + F(\text{Inception}(H[l])), \quad (32)$$

where  $H[l]$  represents the input to the  $l$ th layer,  $\text{Inception}(H[l])$  denotes the output of the Inception module, and  $F$  denotes the residual mapping.

#### 6. NASNetLarge Model

NASNetLarge is a convolutional neural network design that uses the concept of neural architecture search and includes standard and reduction convolution cell blocks. The neural architecture search (NAS) is built to find the optimal CNN architecture automatically using Reinforcement Learning. It is built with the concepts of searching for the best parameter combination for the given search space, output channels, number of layers, and strides. Reinforcement learning is used to find the accuracy of the searched architecture on the given dataset. NASNet emphasizes the automated feature learning process, architecture search, and hyper-parameter optimization. Research has shown that NASNet achieved excellent results on the ImageNet dataset. However, it required high computation power. It is used mainly to search for the best algorithm to achieve the best performance on a specific task [62]. NASNet typically consists of repeated blocks containing convolutional, pooling, batch normalization (BatchNorm), concatenation, and fully connected layers. These blocks are stacked on

top of each other to form the overall architecture. The mathematical expression of batch normalization and concatenation layers is given below:

Given an input tensor  $x$ , the output of a batch normalization layer  $y$  is given by:

$$y = \text{BatchNorm}(x) , \tag{33}$$

where  $\text{BatchNorm}(x)$  is computed by normalizing the activations across each mini-batch. Given  $y_1, y_2, \dots, y_n$  are the inputs to be concatenated, the output of a concatenation layer  $z$  is obtained by concatenating the tensors along the channel dimension as:

$$z = \text{Concat}(y_1, y_2, \dots, y_n) , \tag{34}$$

where  $\text{Concat}(y_1, y_2, \dots, y_n)$  is computed by combining the outputs of multiple preceding layers along the channel dimension.

7. VGG16 Model The Visual Geometry Group (VGG) is known for its simplicity and effectiveness, with a stack of convolutional layers followed by fully connected layers. Mathematical expression for the VGG block can be viewed as:

$$\text{Conv3} - 64 \rightarrow \text{Conv3} - 64 \rightarrow \text{MaxPool2} \rightarrow \text{Conv3} - 128 \rightarrow \text{Conv3} - 128 \rightarrow \text{MaxPool2} \rightarrow \dots \rightarrow \text{FC}, \tag{35}$$

where  $\text{Conv3} - 64$  denotes a convolutional layer with a  $3 \times 3$  kernel and 64 output channels,  $\text{MaxPool2}$  represents max pooling, and  $\text{FC}$  represents fully connected layers. Several versions of VGG Nets have been released, such as VGG16 and VGG19. They mostly have differences, with only the total number of layers included in the network architecture. The VGG16 model is one of the CNN-based models reported as the most successful top image recognition technique with optimized performance due to its acceleration architectural design. It is designed mainly to reduce the parameters used in the CONV layers to improve training time. It contains 16 layers, including  $3 \times 3$  filter layers with a stride 1 and the same padding and max pool layers of a  $2 \times 2$  filter with a stride 2 [58].

8. VGG19 model  
VGG19 is another version of VGG Nets, considered an extension of VGG16. It is built with 19 layers instead of 16 layers used in VGG16. It contains the same architecture as VGG16, with three additional convolutional and max-pooling layers. Research reported that VGG19 had achieved slightly better performance than VGG16 on various image recognition applications; however, it is more computationally expensive due to the more significant number of involved parameters [61].

Consequently, based on the size of processed sign images, which is  $128 \times 128$ , the dimensions of extracted features are given in Table 4 for each CNN transfer learning model utilized as a feature extractor in this study.

**Table 4.** Dimensions of the extracted features from sign images using pre-trained CNN models.

Model	Dimension of Bottleneck Features
	(Filter Size of Height, Filter Size of Width, Number of Channels)
DenseNet121-based Feature Extractor	(4, 4, 1024)
ResNet152-based Feature Extractor	(4, 4, 2048)
MobileNetV2-based Feature Extractor	(4, 4, 1280)
Xception-based Feature Extractor	(4, 4, 2048)
InceptionV3-based Feature Extractor	(4, 4, 2048)
NASNetLarge-based Feature Extractor	(4, 4, 4032)
VGG19-based Feature Extractor	(4, 4, 512)
VGG16-based Feature Extractor	(4, 4, 512)

#### 4.1.4. Classification Model Building

The classification model of the proposed framework is a custom deep neural network (DNN). The DNN is a type of neural network with numerous densely linked layers. It comprises an input layer, several hidden dense layers, and an output layer. A dropout technique can process each hidden layer to avoid the issue of over-fitting. Depending on the nature of the input data, the DNN as a classifier employs a wide range of activation functions. The activation function utilized in the hidden layers of our custom DNN is the rectified linear unit (ReLU). The ReLU is a linear function that, if positive, can directly output the same value of the input; otherwise, it can output zero value. It can train the model quickly and effectively. The input layer of the custom-built DNN model is a 2D global average pooling that takes the extracted features from pre-trained CNN feature extractors. The output layer of the custom DNN model is a fully connected dense layer with a softmax activation function containing several neurons equal to the number of classes in the dataset. The softmax activation function of the output layer gives the probability distribution of each output class value. It can convert the model's outputs into a vector of probabilities over the input classes and produce the class with the highest probability value. The output class of our custom DNN model is a multiclass label of the sign word. Algorithm 1 describes the steps for building the custom DNN model of the proposed framework. In the implementation and experimentation of the algorithm, the number of hidden layers will be set to a small number to achieve efficiency and meet real-time requirements without affecting the model's accuracy.

---

#### Algorithm 1 Custom DNN Architecture Building Algorithm

---

```

Input: input shape;      # the number of neurons in the input layer
# is the shape of extracted bottleneck features from
# pre-trained CNN feature extractors.
# It will be like (filter size of height,
# filter size of width, number of channels).
H;      # the number of hidden layers.
L;      # the number of neurons in the output layer.
Output: M;      # it is the built custom DNN model.
1. Begin
2. Set model M as sequential
3. M ← add(GlobalAveragePooling2D(input_shape))
4. for i = 1 to H
5.     Set N as a suitable number of neurons
6.     M ← add(Dense(N, activation = 'relu'))
7. end for
8. M ← add(Dropout(0.5))
9. M ← add(Dense(L, activation = 'softmax'))
10. return M;
11. End

```

---

#### 4.1.5. Classification Model Training and Hyper-Parameter Tuning

In this step, the custom DNN model was trained on the training set and validated on the validation set. It tuned the hyper-parameters of the custom DNN model as needed to achieve a high-performance result in accuracy and efficiency. We adjusted the batch size, the number of hidden layers, and the number of neurons based on DNN results for the accuracy of the validation set.

Suppose  $F - tri$  is the extracted bottleneck features of the training set and  $F - val$  is the extracted bottleneck features of the validation set, which are extracted from dataset  $D$  of the sign images using each pre-trained CNN feature extractor. The class labels of training and validation sets are denoted by  $L - tri$  and  $L - val$ , respectively. Consider the built DNN model is  $M$ , the loss function is categorical cross-entropy, the optimizer is Adam, and the only metrics are the accuracy of training and validation progress. The goal is training  $M$  on



$F - tri$  and validating on  $F - val$ ; as well as monitoring the accuracy metric for tuning the hyper-parameters of  $M$ . Algorithm 2 gives the pseudo-code for training and validating the Custom DNN model  $M$ .

---

**Algorithm 2** Custom DNN Training Algorithm

---

Input:  $F - tri$ ; # the extracted bottleneck features of the training set.  
 $L - tri$ ; # the class labels of the training set.  
 $F - val$ ; # the extracted bottleneck features of the validation set.  
 $L - val$ ; # the class labels of the validation set.  
 $M$ ; # the custom-built DNN model.  
 $P$ ; # the file path that stores the weights of the best-trained model.  
 $E$ ; # the number of epochs.  
 $B$ ; # the batch of size of training progress.  
Output:  $M$ ; # the trained custom DNN model.  
 $TA, VA$ ; # the training accuracy and validation accuracy.  
 $TL, VL$ ; # the training loss and validation loss.

1. Begin
2.  $loss \leftarrow$  'categorical\_crossentropy'
3.  $optimizer \leftarrow$  'Adam'
4.  $metrics \leftarrow$  ['accuracy']
5.  $M \leftarrow$  compile(loss, optimizer, metrics)
6. Check-Pointer = ModelCheckpoint(filepath =  $P$ , monitor = 'val\_accuracy', save\_best\_only = True, mode = 'max')
7.  $M \leftarrow$  fit( $F - tri, L - tri$ , validation\_data = ( $F - val, L - val$ ), epochs =  $E$ , batch\_size =  $B$ , callbacks = [Check-Pointer], verbose = 1)
8.  $TA, VA \leftarrow$  get-accuracy( $M$ )
9.  $TL, VL \leftarrow$  get-loss( $M$ )
10. plot( $TA, VA$ )
11. plot( $TL, VL$ )
12. return  $M, TA, VA, TL, VL$ ;
13. End

---

For model tuning, Algorithm 2 was executed several times using one hidden layer of 250 neurons and different values of batch size  $B$ , such as 16, 32, 64, and 128, on the features extracted by each pre-trained CNN feature extractor, individually. Based on the highest validation accuracy, the best batch size was selected. After that, using this batch size, Algorithm 2 was used to train the DNN model with a different number of hidden layers  $H$  in Algorithm 1. The trained model  $M$  with the best batch size was evaluated on the extracted features of the test set using all adopted pre-trained CNN feature extractors.

#### 4.1.6. Classification Model Evaluation

Several evaluation metrics are widely used to evaluate the quality of classification models. Evaluation metrics are utilized to assess the performance of built models and monitor their output in the production stage. In this study, the evaluation metrics utilized to assess the trained models were classification accuracy, confusion matrix, precision, recall, and F1-score. These metrics are commonly used for measuring the performance of machine learning classifiers and are described as follows:

- **Classification Accuracy:** It is used to measure the instances of correctly classified predictions. It is the ratio of the number of correctly classified predictions to the total number of predictions. It is calculated by using the equation below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (36)$$

where  $FP$ ,  $FN$ ,  $TP$ , and  $TN$  are the false positive, false negative, true positive, and true negative of instances. It is utilized here because the target classes of the dataset are well-balanced.

- Confusion matrix: Its performance measurement for the machine learning classifiers is used when the output is more than two classes. It is an  $N \times N$  matrix where  $N$  represents the number of classes in the dataset. It is applied to measure the classifier model's performance on the testing dataset. It can compute the number of correct and incorrect predictions made by the model compared with the actual classifications in the test dataset. It is used for measuring precision, recall, and F1-score metrics.
- Precision: The ratio of true positives to all the positives predicted by the model. The following equation is used to calculate the precision metric:

$$Precision = \frac{TP}{TP + FP} \quad (37)$$

- Recall: It is the ratio of true positives to all the positives in the dataset. The following equation is used to calculate the recall metrics:

$$Recall = \frac{TP}{TP + FN} \quad (38)$$

- F1-score: It is a harmonic mean of precision and recall metrics, where it is considered the weighted average of precision and recall with a range of  $[0, 1]$ . The following equation is used to calculate the F1-score metric.

$$F1 - score = 2 \times \frac{(Recall \times Precision)}{(Recall + Precision)} \quad (39)$$

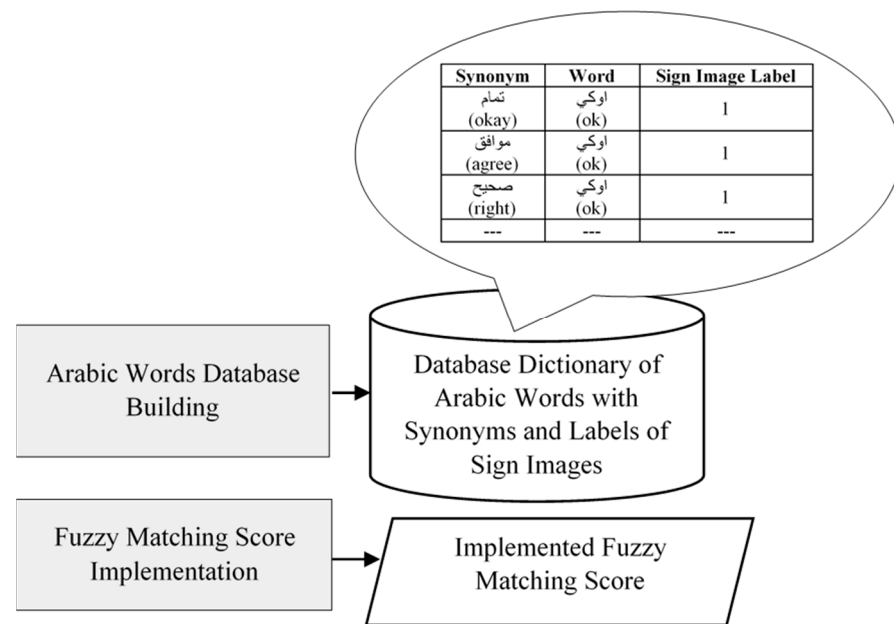
In addition to the above evaluation metrics, a receiver operating characteristic (ROC) curve is plotted to show the trade-off between recall and precision metrics. It can evaluate how the models work in areas with high recall and precision. The ROC curve is usually used in binary classification to compare the performance of trained models; however, it can also be applied to multiclass classification by taking one class as a positive type and the others as a negative type.

#### 4.2. Word-to-Sign Module

The second module in the proposed framework was developed to translate the Arabic input texts from ordinary people into sign images. A fuzzy matching score method was used with a supporting database dictionary of Arabic words for the hand-sign images, as shown in Figure 5 below. This module is designed to solve some existing sign language issues described in the review above. It simplifies the translation process between input text and hand-sign images by connecting the hand-sign images and corresponding word meanings. It resolves issues caused by mistyping, spelling errors, multiple synonyms, and the modified Arabic sign language.

##### 4.2.1. Arabic Words Database Dictionary Building

Ordinary people type text in their local language by using various words compared with standard language words; for example, in Arabic language, words between brackets separated by a comma (حاضر، موافق، تمام، ايوه، نعم) have the same meaning, and the same hand-sign image. In addition, users sometimes misspell words when typing, which cannot be translated directly without correcting these words. For these issues, we developed a support tool using the fuzzy matching score concepts to solve such problems. Thus, the Arabic words database was constructed to support the fuzzy matching score model with the necessary data to solve these issues. The database dictionary uses hand-sign image labels with the associated meaning of the words.



**Figure 5.** Word-to-Sign translation module.

The data dictionary table was designed with three columns to include the word, synonym, and sign image labels. The synonym words column is created with the meanings of similar words and the possible mistyping and spelling errors that could occur during inputs.

#### 4.2.2. Fuzzy String Matching Score Implementation

As mentioned in the review, sign language is considered nonlinear; fuzzy logic is adopted to solve several nonlinear issues, such as sign language. The sign translation system proposed in this study is meant to translate between both deaf and ordinary people. Thus, we implemented a novel model to obtain text from ordinal users and convert it into appropriate sign language images. A fuzzy matching model was used in the prototype to determine appropriate signs for the text or phonetics that have different words with the same meaning, such as “Yes”, “Ok”, and “Alright”. As discussed previously, different words can be used due to the diverse regions with the same meaning. Thus, a fuzzy matching score was implemented by using the FuzzyWuzzy library. The Arabic database dictionary includes different words used for the same word meaning, which can minimize the dataset domain, improve the system performance, and simplify the translation process. The fuzzy matching model used the database Arabic dictionary to translate the input word from ordinary people into corresponding sign images.

The fuzzy matching score is a technique that is used to identify items in a dataset based on their similarities. It compares strings and assigns a score value to each string based on their similarity. The fuzzy matching operator calculates the Levenshtein distance between strings and assigns a score between 0 and 1. Fuzzy matching is considered an efficient search technique that uses a set of fuzzy rules with some similarity for string comparison. We use the formula of the Levenshtein distance between two strings  $a$  and  $b$ , which is given in Equation (27), to obtain the semantics and synonyms of the same Arabic words for retrieving the corresponding Arabic sign language images. Fuzzy matching is an essential component of semantic search since it aids in locating the closest match to a text inside a context. It serves as the foundation for semantic analysis. Semantic search comes in various styles, all of which are based on term contextualization and the intent behind it. It works better than simple text matching and depends on more than a deterministic, binary-based information retrieval mechanism. Algorithm 3 represents the Levenshtein distance method using a recurrence relation. Where  $a_i$  is a word/character in the string  $a$ ,

and  $b_j$  is a word/character in string  $b$ , the  $equal(a_i, b_j)$  is a function that returns zero if the two words/characters are equal and one otherwise.

---

**Algorithm 3** Levenshtein Distance Represented by the Recurrence Relation

---

```

1. Begin
2. for each  $i$  from 0 to  $|a|$ 
3.   for each  $j$  from 0 to  $|b|$ 
4.      $lev(0, 0) = 0;$ 
5.      $lev(i, 0) = i;$ 
6.      $lev(0, j) = j;$ 
7.      $lev(i, j) = \min [lev(i - 1; j) + 1;$ 
8.        $lev(i, j - 1) + 1;$ 
9.        $lev(i - 1, j - 1) + equal(a_i, b_j)]$ 
10. End

```

---

The fuzzy score matching model does not require any evaluation because it depends mainly on the distance between the words and the dictionary. Thus, the model retrieves the label from a list of similar input words based on the nearest matching of the string.

## 5. Experimental Results and Discussion

The prototype of the proposed framework was implemented using Python programming language on the Google Compute Engine backend with a T4 GPU for the hardware accelerator, 12.7 GB of system RAM, and 78.2 GB of disk RAM. We evaluated each module of the developed framework separately. The fuzzy string matching score in the word-to-sign module was mathematically validated by computing the distance between the input and associative dictionary words. On the other hand, the sign-to-word module was validated using several experiments conducted on the dataset images using the eight adopted pre-trained CNN models of feature extraction. The custom DNN model was trained on the extracted features at 200 epochs during the experiments. The DNN model was initially built with a 2D global average pooling, a hidden layer of 250 units with the rectified linear unit (ReLU), a dropout ratio of 0.5, and a learning rate of 0.001. The experiment was conducted with a different number of batch sizes, as mentioned in Section 4.1.5. The model's weights in the training process were updated using Adam's optimizer. In addition, two individual experiments were conducted to select the suitable batch size and several hidden layer parameters. The first experiment was performed to find the batch size ideal for the given dataset. Table 5 shows the validation accuracy of the custom DNN model trained on the extracted features using different parameters of batch size values.

**Table 5.** Validation accuracy of the pre-trained CNN feature extractor-based Custom DNN model based on a splitting procedure (80:20) using different batch sizes.

Model	Batch Size			
	16	32	64	128
DenseNet121-based Custom DNN	99.022%	99.022%	<b>99.200%</b>	99.111%
ResNet152-based Custom DNN	63.733%	<b>65.867%</b>	65.244%	62.489%
MobileNetV2-based Custom DNN	98.222%	98.133%	<b>98.489%</b>	98.311%
Xception-based Custom DNN	<b>97.511%</b>	97.422%	97.244%	<b>97.511%</b>
InceptionV3-based Custom DNN	95.378%	95.733%	95.644%	<b>96.000%</b>
NASNetLarge-based Custom DNN	95.733%	95.822%	<b>95.911%</b>	95.822%
VGG19-based Custom DNN	98.311%	<b>98.489%</b>	<b>98.489%</b>	98.222%
VGG16-based Custom DNN	<b>99.022%</b>	98.844%	98.933%	98.844%

As shown in Table 5, the DenseNet121-based Custom DNN model achieved the highest validation accuracy, 99.2%, highlighted with a batch size of 64. Moreover, it shows that most of the models attained high accuracy results, as highlighted in bold font, at batch

size 64 compared with the other batch size values. In addition, the second-best model, the VGG16-based Custom DNN, also reached 99.022% validation accuracy at a batch size of 16. In contrast to the ResNet152-based Custom DNN model, it obtained the lowest validation accuracy results at all batch sizes. Based on the validation accuracy results shown in Table 5, batch size 64 was selected for the proposed framework's classification model. The second experiment determined the suitable number of DNN's hidden layer parameters. In this experiment, the model was trained using the selected batch size and evaluated on the extracted features of the test set. Table 6 presents the testing accuracy results of the pre-trained CNN feature extractor-based Custom DNN model using a different number of hidden layers and neurons.

**Table 6.** Testing accuracy of pre-trained CNN feature extractor-based Custom DNN model based on a splitting procedure (80:20) using different numbers of hidden layers and neurons.

Model	Number of Hidden Layers and Neurons		
	(256)	(256, 128)	(256, 128, 64)
DenseNet121-based Custom DNN	98.15%	<b>98.36%</b>	98.15%
ResNet152-based Custom DNN	47.23%	63.44%	<b>66.50%</b>
MobileNetV2-based Custom DNN	<b>98.51%</b>	98.08%	97.58%
Xception-based Custom DNN	96.44%	<b>97.16%</b>	<b>97.16%</b>
InceptionV3-based Custom DNN	<b>94.31%</b>	94.24%	93.46%
NASNetLarge-based Custom DNN	<b>96.23%</b>	95.66%	96.09%
VGG19-based Custom DNN	<b>98.22%</b>	97.80%	98.01%
VGG16-based Custom DNN	<b>98.65%</b>	<b>98.65%</b>	98.44%

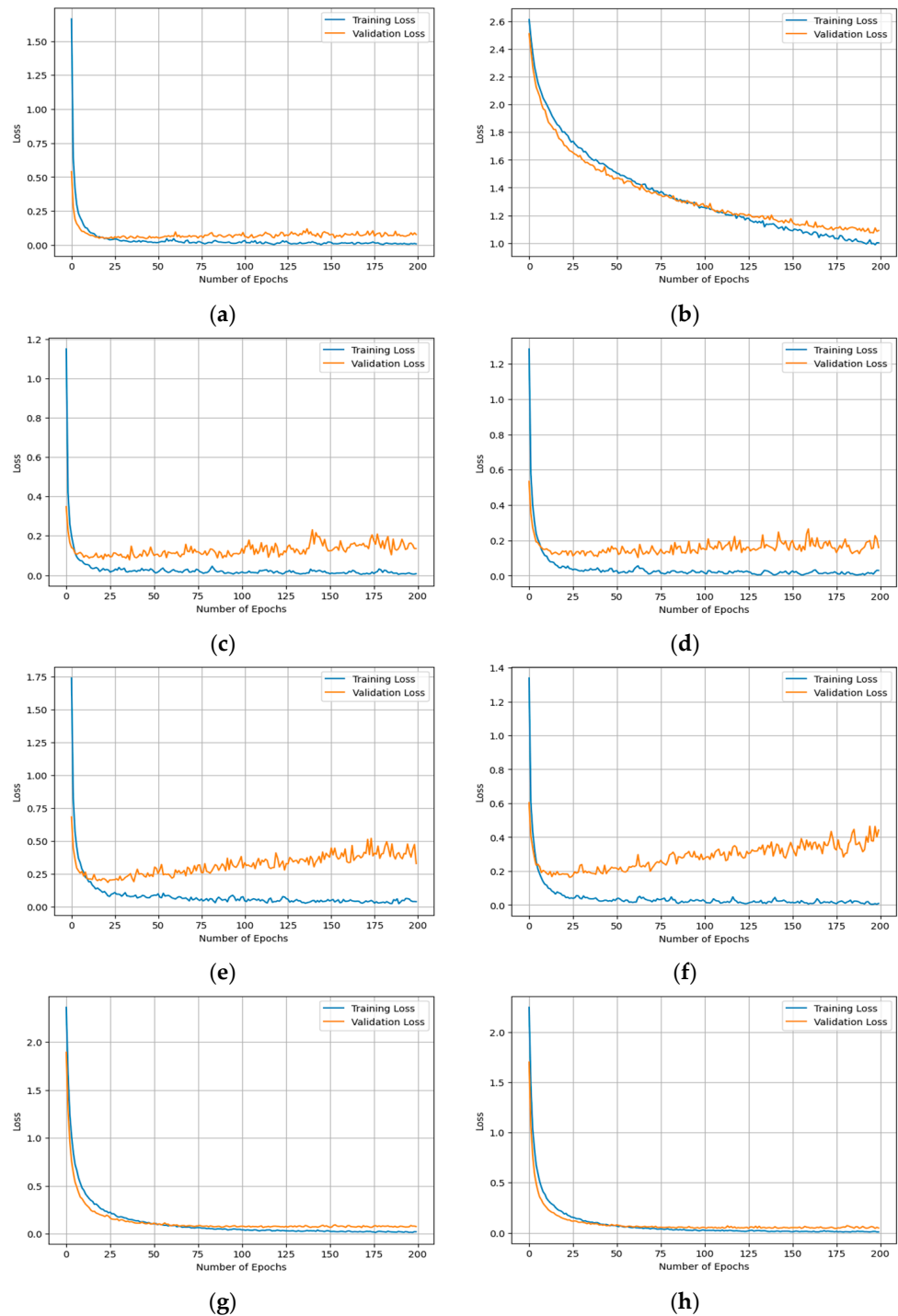
As shown in Table 6, the VGG16-based Custom DNN model attained the highest testing accuracy, 98.65%, using one hidden layer of 256 neurons and two hidden layers of 256 and 128, respectively. Furthermore, most models achieved high accuracy, as highlighted in bold font, when applied to the hidden layer of 256 neurons. Therefore, we have experimentally found these are the most appropriate values of hyperparameters for our sign-to-word module. According to the accuracy results in Tables 5 and 6, the Custom DNN architecture with a batch size of 64 and a hidden layer of 256 neurons was selected to build the Arabic word sign language classifier. The amount of loss and accuracy for each pre-trained CNN feature extractor-based model on training and validation sets of first splitting procedure is shown in Figures 6 and 7.

As shown in Figures 6 and 7, they can help reveal more details about the training progress of the proposed DNN model in terms of over-fitting and under-fitting issues. We can see that the gaps between the training and validation curves of the Custom DNN model using the DenseNet121, VGG16, and VGG19 feature extractors are very small. However, as shown in Figures 6h and 7h, the gap of training and validation loss and accuracy for the VGG16-based model is as small as possible, which indicates its performance quality in the training phase and means that it achieved the best performance result in the testing phase. In contrast, the poor performance of the ResNet152-based model in Figures 6b and 7b, which reveals its training weakness due to the complexity of ResNet152, generating a set of features which makes the partial derivative values of the loss function to the models' weights minimal. Additionally, we see that the gaps in training and validation loss and accuracy for the other models in Figures 6c–f and 7c–f are high, meaning they suffer from an over-fitting problem.

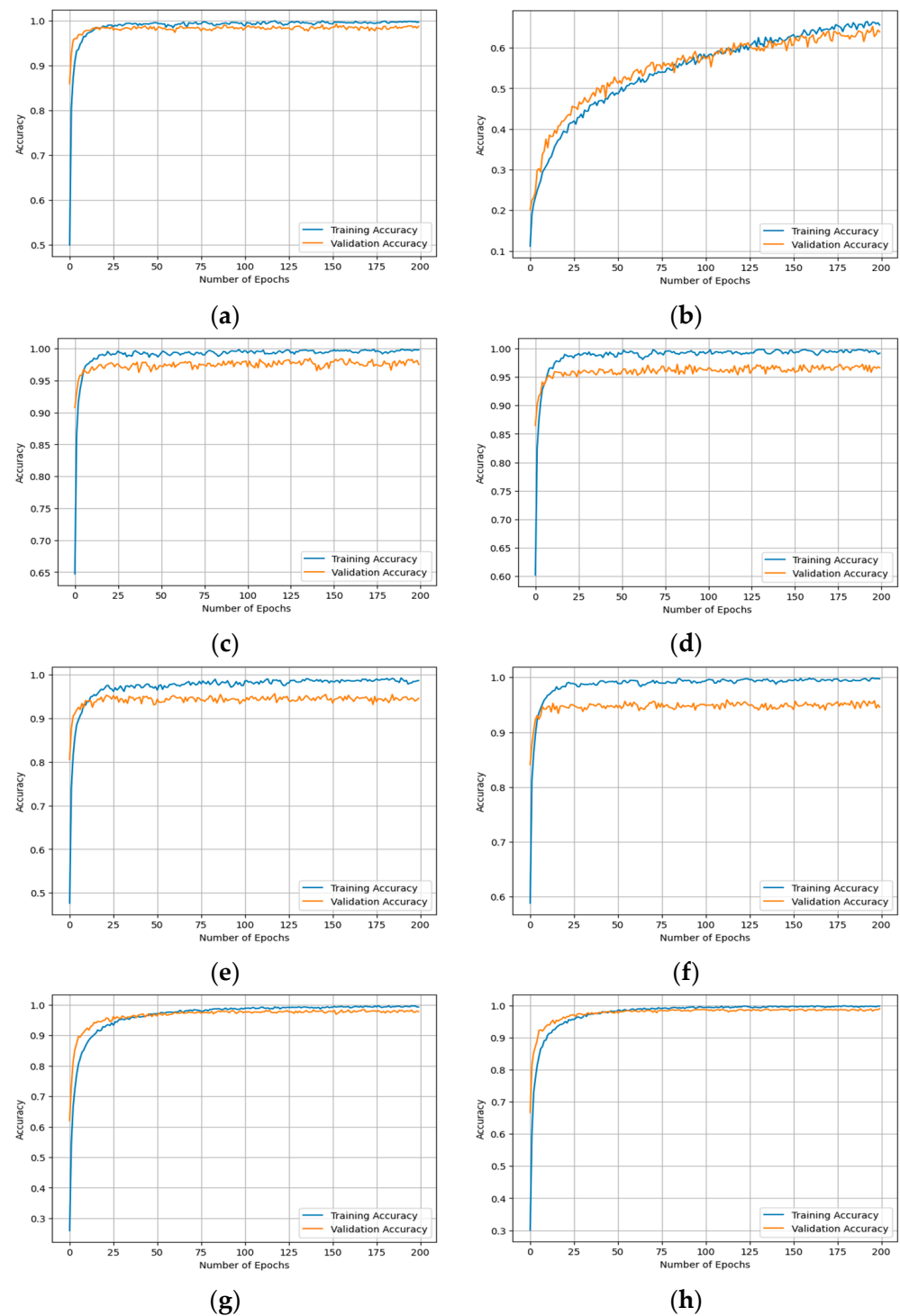
The confusion matrix was also used to measure the performance of the pre-trained CNN feature extractor-based Custom DNN model to evaluate the confidence level. Figures 8–15 show the confusion matrices of the pre-trained feature extractor-based Custom DNN model on the test set of this study. The confusion matrix of the VGG16 feature extractor in Figure 15 shows that the model accuracy is nearly identical in the diagonal elements, which indicates a higher classification accuracy rate. From the confusion matrices in Figures 8–15, the other evaluation metrics, such as precision, recall, and F1-score,



were computed for more clarification. Tables 7–14 below illustrate the results of these measurements for the model using adopted pre-trained CNN feature extractors. Moreover, in Figures 16 and 17, we visualize and compare the accuracy, precision, recall, and F1-score results for the utilized models by using all pre-trained CNN feature extractors, the results of which are summarized in Tables 7–14. The results are interpreted in more detail below.



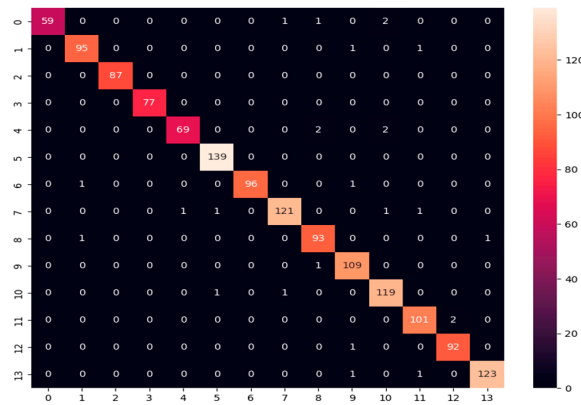
**Figure 6.** The amount of training and validation loss for the Custom DNN model based on a splitting procedure (80:20): (a) using the DenseNet121 feature extractor, (b) using the ResNet152 feature extractor, (c) using the MobileNetV2 feature extractor, (d) using the Xception feature extractor, (e) using the InceptionV3 feature extractor, (f) using the NASNetLarge feature extractor, (g) using the VGG19 feature extractor, and (h) using the VGG16 feature extractor.



**Figure 7.** The amount of training and validation accuracy for the Custom DNN model based on a splitting procedure (80:20): (a) using the DenseNet121 feature extractor, (b) using the ResNet152 feature extractor, (c) using the MobileNetV2 feature extractor, (d) using the Xception feature extractor, (e) using the InceptionV3 feature extractor, (f) using the NASNetLarge feature extractor, (g) using the VGG19 feature extractor, and (h) using the VGG16 feature extractor.

The confusion matrix results illustrated in Figure 8 for the DenseNet121 model represent an identical prediction value for most classes with false classification for some instances, as clearly shown by comparing the diagonal values of a matrix with the total number of instances for each class in the test set. For example, class 8 has 93 instances

classified correctly from 95 instances in the test set, with only two misclassified instances. The higher misclassification rate among all classes is four instances only, as shown in class numbers 0, 4, and 7. Additionally, the class-wise accuracy, precision, recall, and F1-score were computed for each class, as shown in Table 7. The results in Table 7 show that the precision value of classes for the model is between 0.9588, the lowest in class 8, and 1.0, the highest in classes 2, 3, and 6.



**Figure 8.** Confusion matrix of the DenseNet121-based Custom DNN model on the test set using an 80:20 splitting procedure.

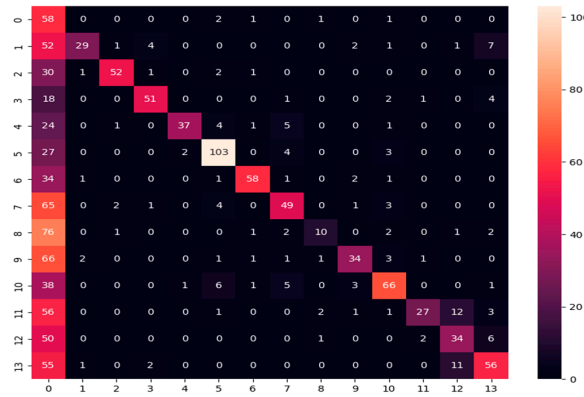
**Table 7.** Precision, recall, and F1-score results of the DenseNet121-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class No.	Precision	Recall	F1-Score
0	1.0000	0.9365	0.9672
1	0.9794	0.9794	0.9794
2	1.0000	1.0000	1.0000
3	1.0000	1.0000	1.0000
4	0.9857	0.9452	0.9650
5	0.9858	1.0000	0.9929
6	1.0000	0.9796	0.9897
7	0.9837	0.9680	0.9758
8	0.9588	0.9789	0.9688
9	0.9646	0.9909	0.9776
10	0.9597	0.9835	0.9714
11	0.9712	0.9806	0.9758
12	0.9787	0.9892	0.9840
13	0.9919	0.9840	0.9880
Accuracy		98.15%	
Macro avg.	0.9828	0.9797	0.9811
Weighted avg.	0.9817	0.9815	0.9815

Similarly, the recall value of classes varies between [0.9365 and 1.0], and the F1-score varies between [0.9650 and 1.0]. In addition, the macro-average and weighted average as global metrics were computed to evaluate the overall model’s performance over the entire dataset. As shown in Table 7, the overall performance for DenseNet121 was 98.28% in macro precision, 97.97% in macro recall, 98.11% in macro F1-score, and the overall model accuracy was 98.15%. Furthermore, the weighted average gives broadly similar results to the macro-average.

The confusion matrix results illustrated in Figure 9 for the ResNet152 model show a higher misclassification for most classes, as shown by the F1-score in the last column of Table 8. The ResNet152 model had a 47.23% accuracy rate, as given in Table 8, where the other metrics were precision = 0.75, recall = 0.48, and F1-score = 0.52. These results are unsuitable for application to such a problem domain. They may have occurred due to the complexity of ResNet152, which generates a set of features that minimize the partial derivative values

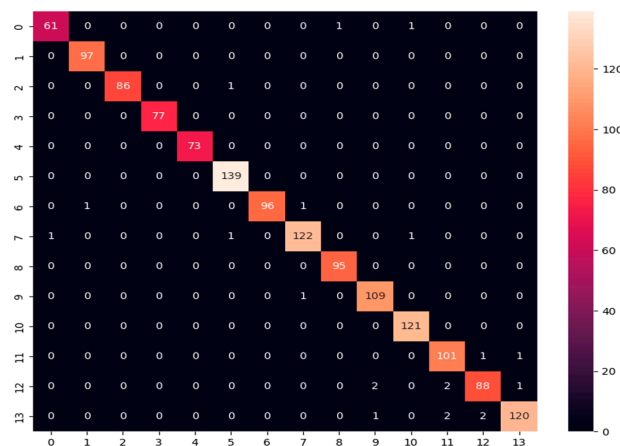
of the loss function to the models' weights. Extra evaluation with reconfiguration of its structure is suggested for improving the performance of the ResNet152. However, the other models achieved excellent evaluation results for our proposed framework.



**Figure 9.** Confusion matrix of the ResNet152-based Custom DNN model on the test set using an 80:20 splitting procedure.

**Table 8.** Precision, recall, and F1-score results of the ResNet152-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class Number	Precision	Recall	F1-Score
0	0.0894	0.9206	0.1629
1	0.8529	0.2990	0.4427
2	0.9123	0.5977	0.7222
3	0.8644	0.6623	0.7500
4	0.9250	0.5068	0.6549
5	0.8306	0.7410	0.7833
6	0.9062	0.5918	0.7160
7	0.7206	0.3920	0.5078
8	0.6667	0.1053	0.1818
9	0.7907	0.3091	0.4444
10	0.7857	0.5455	0.6439
11	0.8710	0.2621	0.4030
12	0.5763	0.3656	0.4474
13	0.7089	0.4480	0.5490
Accuracy		47.23%	
Macro avg.	0.7500	0.4819	0.5292
Weighted avg.	0.7635	0.4723	0.5405

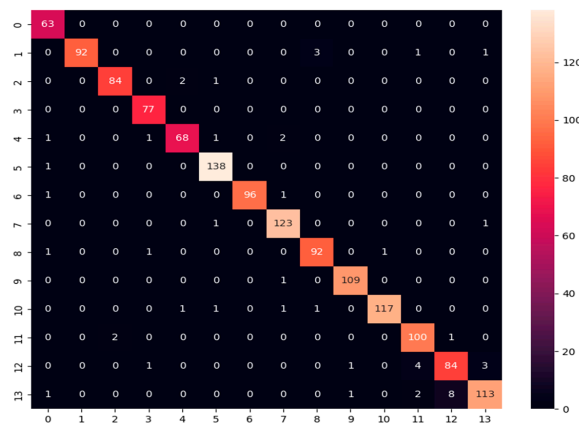


**Figure 10.** Confusion matrix of the MobileNetV2-based Custom DNN model on the test set using an 80:20 splitting procedure.

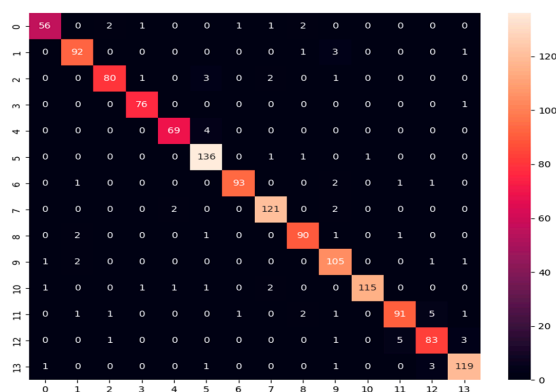
**Table 9.** Precision, recall, and F1-score results of the MobileNetV2-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class Number	Precision	Recall	F1-Score
0	0.9839	0.9683	0.9760
1	0.9898	1.0000	0.9949
2	1.0000	0.9885	0.9942
3	1.0000	1.0000	1.0000
4	1.0000	1.0000	1.0000
5	0.9858	1.0000	0.9929
6	1.0000	0.9796	0.9897
7	0.9839	0.9760	0.9799
8	0.9896	1.0000	0.9948
9	0.9732	0.9909	0.9820
10	0.9837	1.0000	0.9918
11	0.9619	0.9806	0.9712
12	0.9670	0.9462	0.9565
13	0.9836	0.9600	0.9717
Accuracy		98.51%	
Macro avg.	0.9859	0.9850	0.9854
Weighted avg.	0.9851	0.9851	0.9850

The confusion matrix results illustrated in Figure 10 for the MobileNetV2 model show excellent performance with false misclassifications, as appears in the diagonal value of the matrix. Furthermore, Table 9 shows that the MobileNetV2 model had competitive performance results for accuracy of 98.51%, precision 98.5%, recall 98.5%, and F1-score 98.5%.



**Figure 11.** Confusion matrix of the Xception-based Custom DNN model on the test set using an 80:20 splitting procedure.



**Figure 12.** Confusion matrix of the InceptionV3-based Custom DNN model on the test set using an 80:20 splitting procedure.

**Table 10.** Precision, recall, and F1-score results of the Xception-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class Number	Precision	Recall	F1-Score
0	0.9265	1.0000	0.9618
1	1.0000	0.9485	0.9735
2	0.9767	0.9655	0.9711
3	0.9625	1.0000	0.9809
4	0.9577	0.9315	0.9444
5	0.9718	0.9928	0.9822
6	1.0000	0.9796	0.9897
7	0.9609	0.9840	0.9723
8	0.9583	0.9684	0.9634
9	0.9820	0.9909	0.9864
10	0.9915	0.9669	0.9791
11	0.9346	0.9709	0.9524
12	0.9032	0.9032	0.9032
13	0.9576	0.9040	0.9300
Accuracy		96.44%	
Macro avg.	0.9631	0.9647	0.9636
Weighted avg.	0.9648	0.9644	0.9644

**Table 11.** Precision, recall, and F1-score results of the InceptionV3-based Custom DNN model on the test set using an 80:20 splitting procedure.

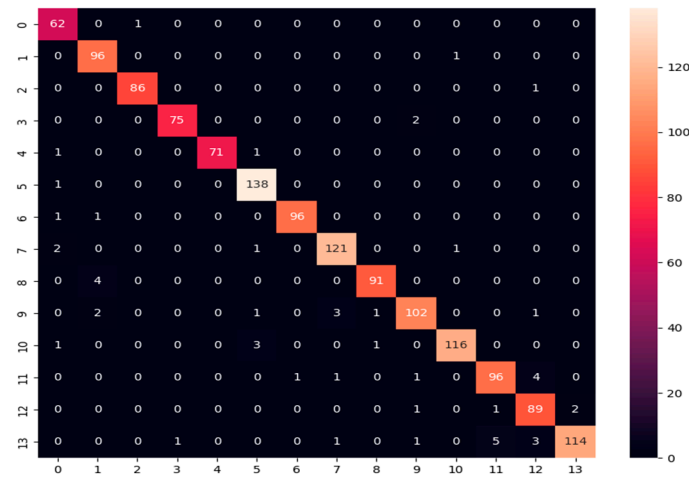
Class Number	Precision	Recall	F1-Score
0	0.9492	0.8889	0.9180
1	0.9388	0.9485	0.9436
2	0.9524	0.9195	0.9357
3	0.9620	0.9870	0.9744
4	0.9583	0.9452	0.9517
5	0.9315	0.9784	0.9544
6	0.9789	0.9490	0.9637
7	0.9528	0.9680	0.9603
8	0.9375	0.9474	0.9424
9	0.8974	0.9545	0.9251
10	0.9914	0.9504	0.9705
11	0.9286	0.8835	0.9055
12	0.8925	0.8925	0.8925
13	0.9444	0.9520	0.9482
Accuracy		94.31%	
Macro avg.	0.9440	0.9403	0.9419
Weighted avg.	0.9436	0.9431	0.9430

By comparing the confusion matrix results shown in Figure 11 for the Xception model with the total instance for each class, a remarkable performance is obtained with a low misclassification rate, as given in the diagonal value of the matrix. Furthermore, Table 10 shows that the MobileNetV2 model attained 96.44%, 96.3%, 96.5%, and 96.4% for accuracy, precision, recall, and F1-score, respectively.

The confusion matrix results shown in Figure 12 for the InceptionV3 model showed an excellent performance in terms of false misclassifications, as shown in the diagonal values of the confusion matrix. Furthermore, Table 11 shows that the InceptionV3 model had an accuracy rate of 94.3%, precision = 94.4%, recall = 94%, and F1-score = 94.2%.

The confusion matrix results shown in Figure 13 for the NASNet model show competitive performance with a few false misclassifications, as shown in the diagonal values of the confusion matrix. Furthermore, Table 12 indicates that NASNetLarge model had an accuracy rate of 96.2%, precision = 96.3%, recall = 96.4%, and F1-score = 96.3%.

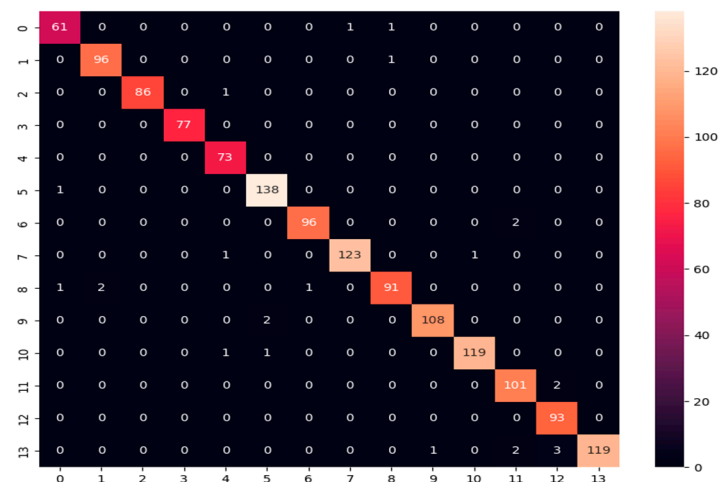




**Figure 13.** Confusion matrix of the NASNetLarge-based Custom DNN model on the test set using an 80:20 splitting procedure.

**Table 12.** Precision, recall, and F1-score results of the NASNetLarge-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class Number	Precision	Recall	F1-Score
0	0.9118	0.9841	0.9466
1	0.9320	0.9897	0.9600
2	0.9885	0.9885	0.9885
3	0.9868	0.9740	0.9804
4	1.0000	0.9726	0.9861
5	0.9583	0.9928	0.9753
6	0.9897	0.9796	0.9846
7	0.9603	0.9680	0.9641
8	0.9785	0.9579	0.9681
9	0.9533	0.9273	0.9401
10	0.9831	0.9587	0.9707
11	0.9412	0.9320	0.9366
12	0.9082	0.9570	0.9319
13	0.9828	0.9120	0.9461
Accuracy		96.23%	
Macro avg.	0.9625	0.9639	0.9628
Weighted avg.	0.9631	0.9623	0.9623



**Figure 14.** Confusion matrix of the VGG19-based Custom DNN model on the test set using an 80:20 splitting procedure.

The confusion matrix results shown in Figure 14 for the VGG19 model showed an excellent performance with false misclassifications, as shown in the diagonal values of the confusion matrix. Furthermore, Table 13 indicates that the VGG19 model had performance results of 98.22%, 98.2%, 98.3%, and 98.2% in terms of accuracy, precision, recall, and F1-score, respectively.

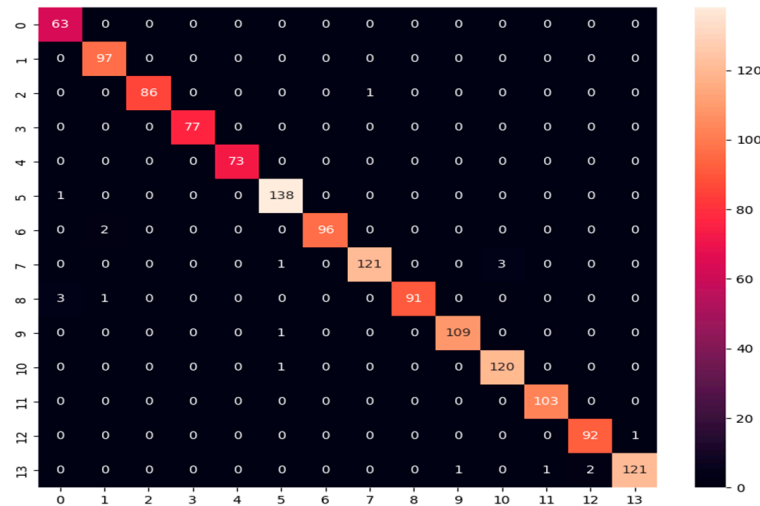


Figure 15. Confusion matrix of the VGG16-based Custom DNN model on the test set using an 80:20 splitting procedure.

Table 13. Precision, recall, and F1-score results of the VGG19-based Custom DNN model on the test set using an 80:20 splitting procedure.

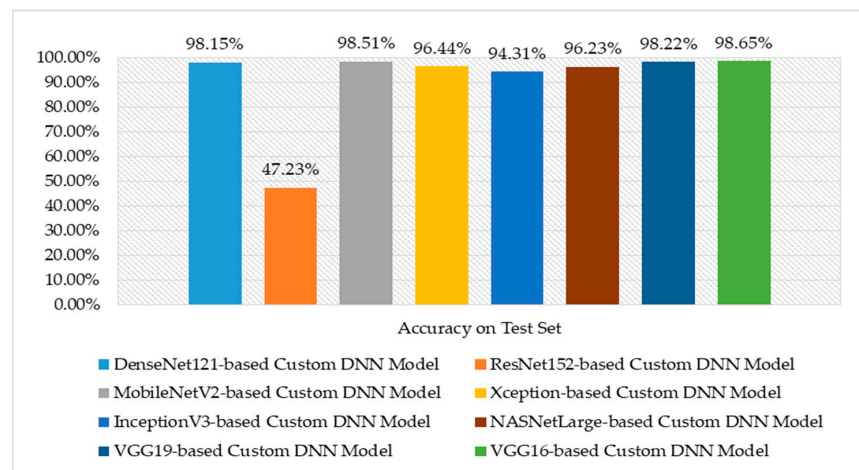
Class Number	Precision	Recall	F1-Score
0	0.9683	0.9683	0.9683
1	0.9796	0.9897	0.9846
2	1.0000	0.9885	0.9942
3	1.0000	1.0000	1.0000
4	0.9605	1.0000	0.9799
5	0.9787	0.9928	0.9857
6	0.9897	0.9796	0.9846
7	0.9919	0.9840	0.9880
8	0.9785	0.9579	0.9681
9	0.9908	0.9818	0.9863
10	0.9917	0.9835	0.9876
11	0.9619	0.9806	0.9712
12	0.9490	1.0000	0.9738
13	1.0000	0.9520	0.9754
Accuracy		98.22%	
Macro avg.	0.9815	0.9828	0.9820
Weighted avg.	0.9825	0.9822	0.9822

The confusion matrix results shown in Figure 15 for the VGG16 model showed an excellent performance in terms of false misclassifications, as shown in the confusion matrix’s diagonal values. Furthermore, Table 14 showed that the VGG16 model had an accuracy rate of 98.65%, precision = 98.63%, recall = 98.76%, and F1-score = 98.68%.

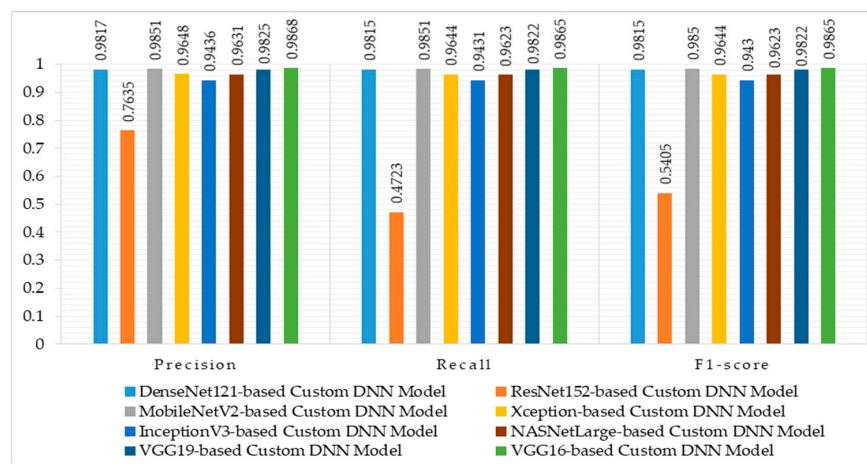
For analyzing and comparing the performance of the adopted feature extraction-based methods, Table 15 compares evaluation metrics using an 80:20 splitting procedure for utilized pre-trained CNN feature extractors combined with the DNN classification model.

**Table 14.** Precision, recall, and F1-score results of the VGG16-based Custom DNN model on the test set using an 80:20 splitting procedure.

Class Number	Precision	Recall	F1-Score
0	0.9403	1.0000	0.9692
1	0.9700	1.0000	0.9848
2	1.0000	0.9885	0.9942
3	1.0000	1.0000	1.0000
4	1.0000	1.0000	1.0000
5	0.9787	0.9928	0.9857
6	1.0000	0.9796	0.9897
7	0.9918	0.9680	0.9798
8	1.0000	0.9579	0.9785
9	0.9909	0.9909	0.9909
10	0.9756	0.9917	0.9836
11	0.9904	1.0000	0.9952
12	0.9787	0.9892	0.9840
13	0.9918	0.9680	0.9798
Accuracy		98.65%	
Macro avg.	0.9863	0.9876	0.9868
Weighted avg.	0.9868	0.9865	0.9865



**Figure 16.** Accuracy results of pre-trained feature extractor-based Custom DNN models on the test set using an 80:20 splitting procedure.



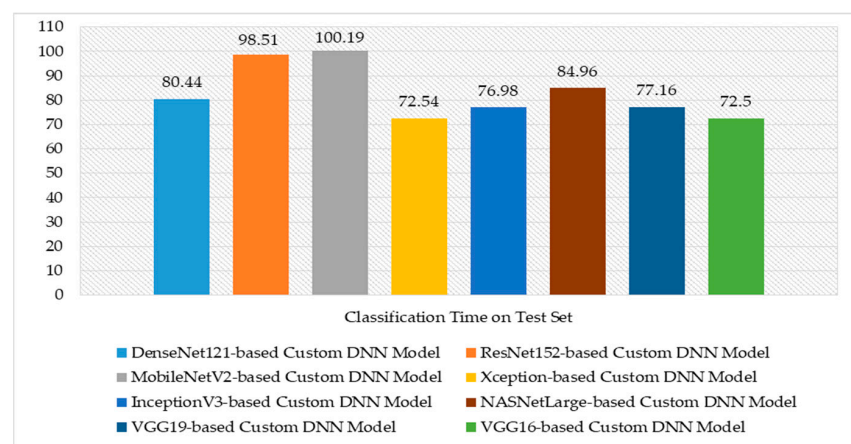
**Figure 17.** Precision, recall, and F1-score results of pre-trained feature extractor-based Custom DNN model on the test set using an 80:20 splitting procedure.

**Table 15.** Comparison of performance results for utilized pre-trained CNN feature extractors combined with the DNN classification model on the test set using an 80:20 splitting procedure.

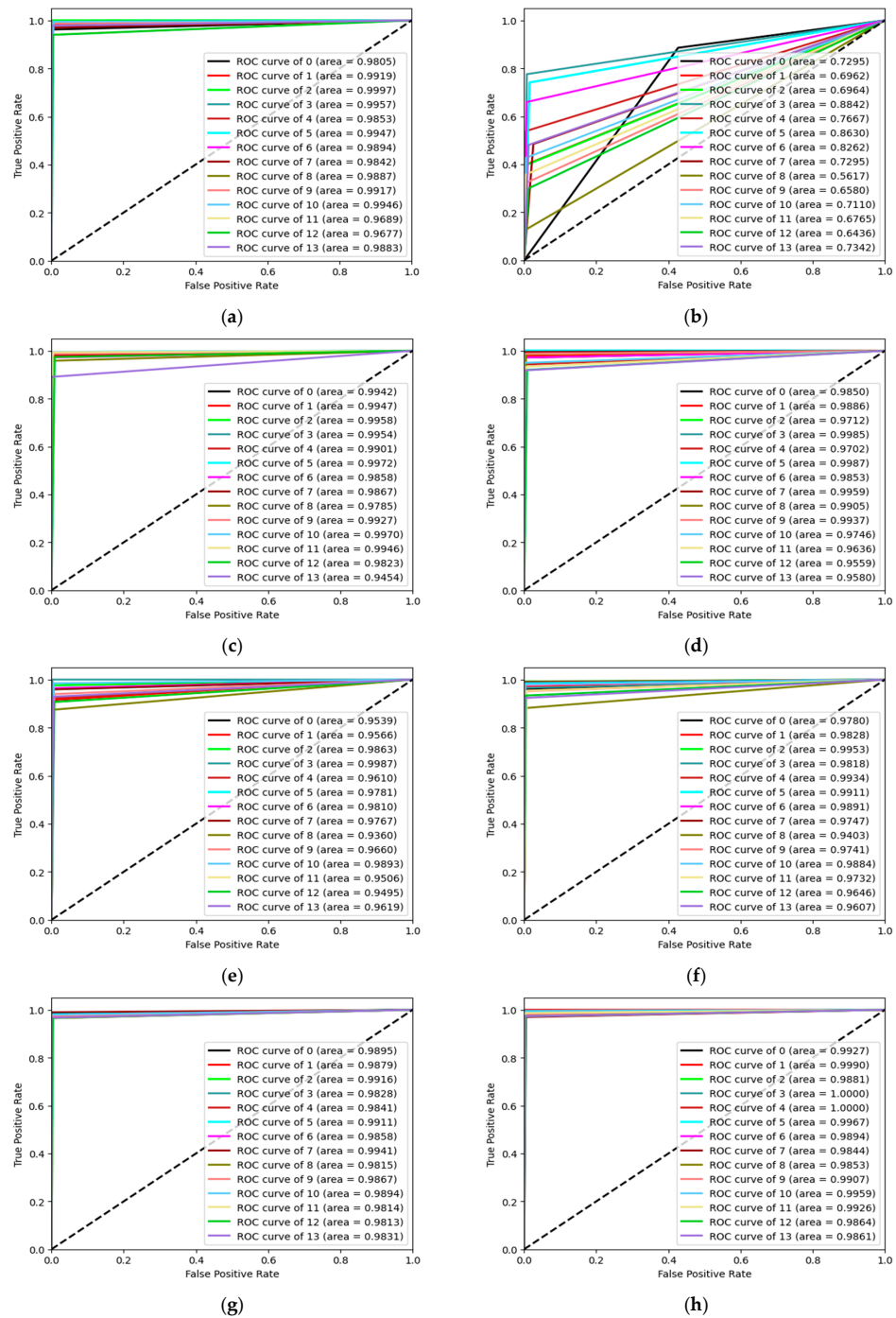
Model	Accuracy	Precision	Recall	F1-Score
DenseNet121	98.15%	0.9828	0.9797	0.9811
ResNet152	47.23%	0.7500	0.4819	0.5292
MobileNetV2	98.51%	0.9859	0.9850	0.9854
Xception	96.44%	0.9631	0.9647	0.9636
InceptionV3	94.31%	0.9440	0.9403	0.9419
NASNetLarge	96.23%	0.9625	0.9639	0.9628
VGG19	98.22%	0.9815	0.9828	0.9820
VGG16	98.65%	0.9863	0.9876	0.9868

As shown in Table 15, the VGG16, MobileNetV2, VGG19, and DenseNet121 models obtained the highest performance results. They achieved an accuracy rate of 98.65%, 98.51%, 98.22%, and 98.15%, respectively. Furthermore, the Xception model, NASNetLarge model, and InceptionV3 model attained competitive performance results with accuracy rates of 96.44%, 96.23%, and 94.31%, respectively. However, the ResNet152 model failed to classify the test set well, whereby it obtained the lowest accuracy rate of 47.23%. The performance results of other metrics, such as precision, recall, and F1-score, are also illustrated in Table 15, confirming the consistency of achieved performance for each CNN model. The accuracy rate, precision, recall, and F1-score results are visualized in graph charts in Figures 16 and 17 for easy comparison between the CNN models.

Finally, to measure the efficiency of the proposed DNN models, the average computation time in seconds for classifying all test set instances was calculated as illustrated in Figure 18. It shows that the VGG16-based Custom DNN model and Xception-Based Custom DNN achieved the lowest computation time, confirming their efficiency for the real-time translation framework of Arabic word sign language into appropriate Arabic text words. Also, Figure 18 shows that the InceptionV3 model, VGG19 model, and DenseNet121 model achieved good efficiency in terms of computation time, while the NASNetLarge model, ResNet152 model, and MobileNetV2 model achieved the lowest efficiency of computation times.

**Figure 18.** Average classification time of pre-trained feature extractor-based Custom DNN model on the test set using an 80:20 splitting procedure.

To generalize and confirm the performance of developed models using the first splitting procedure (80:20), we conducted another experiment using the second splitting procedure (70:30). Table 16 gives a comparison of performance results on the test set using an 70:30 splitting procedure for pre-trained CNN feature extractors combined with the DNN classification model. Figure 19 illustrates the ROC curves for each class label of every built model.



**Figure 19.** ROC curves of pre-trained feature extractor-based Custom DNN model on the test set using an 70:30 splitting procedure: (a) ROC curve of the DenseNet121-based Custom DNN model, (b) ROC curve of the ResNet152-based Custom DNN model, (c) ROC curve of the MobileNetV-based Custom DNN model, (d) ROC curve of the Xception-based Custom DNN model, (e) ROC curve of the InceptionV3-based Custom DNN model, (f) ROC curve of the NASNetLarge-based Custom DNN model, (g) ROC curve of the VGG19-based Custom DNN model, and (h) ROC curve of the VGG16-based Custom DNN model.

Table 16 shows that the VGG16, VGG19, MobileNetV2, and DenseNet121 models achieved the highest performance results. They attained 98.44%, 97.53%, 97.63%, and 97.68% in terms of classification performance, respectively. Moreover, the Xception, NASNetLarge, and InceptionV3 models performed well with accuracy rates of 96.44%, 95.78%, and 94.12%, respectively. However, the performance result of the ResNet152 model was the lowest,

achieving an accuracy rate of 48.36%. The performance results of other metrics, such as precision, recall, and F1-score for each model, are also given in Table 16. For more analysis of classification results, ROC curves in Figure 19 were used as an essential measure to assess the overall quality of the models and determine which one performed best. They calculate the FPR and TPR for each model at different thresholds, in which the FPR values are given on the  $x$ -axis and the values of TPR are set on the  $y$ -axis. Each point on the ROC curve corresponds to a pair of FPR and TPR values for each particular threshold value. From the ROC curves in Figure 19g,h, we notice that VGG19 and VGG16 feature extractors combined with our Custom DNN model achieved outstanding classification results concerning FPR and TPR values for each class label. This confirms the quality and effectiveness of the sign-to-word translation module in recognizing the Arabic sign images of the test set.

**Table 16.** Comparison of performance results for utilized pre-trained CNN feature extractors combined with the DNN classification model on the test set using a 70:30 splitting procedure.

Model	Accuracy	Precision	Recall	F1-Score
DenseNet121	97.68%	0.9771	0.9768	0.9768
ResNet152	48.36%	0.7612	0.4836	0.5486
MobileNetV2	97.63%	0.9773	0.9763	0.9763
Xception	96.44%	0.9650	0.9644	0.9645
InceptionV3	94.12%	0.9417	0.9412	0.9411
NASNetLarge	95.78%	0.9586	0.9578	0.9578
VGG19	97.53%	0.9756	0.9753	0.9754
VGG16	98.44%	0.9848	0.9844	0.9844

In general, the proposed framework of our study has a number of advantages and shows an outstanding accuracy result compared with the previous studies described in the literature and listed in Table 17. The one-directional translation of Arabic sign language is the central research gap of the existing models and methods. They focused on translating the Arabic sign images into their corresponding words or letters. In addition, current state-of-the-art studies have several limitations, such as low accuracy rates, high computational resources, and limited evaluation metrics. Hence, this study was conducted to fill in the gaps and limitations of the previous work.

**Table 17.** A comparative analysis of proposed work compared to the state-of-the-art studies.

[Ref.] (Year)	Model Advantages and Disadvantages	Sign Letters or Words	Accuracy	Is Efficiency Measured?	Is it Bidirectional?
	Pros:				
	- Proposed a simple vision-based system using a CNN with two convolution layers and two maximum pooling layers for recognizing the Arabic hand-sign letters.				
	- Generated Arabic speech for each recognized Arabic hand-sign letter.				
	- Applied data preprocessing to enhance the representation of sign images.				
[32] (2020)	Cons:	Letters	90%	Not Measured	No
	- Training the CNN model from scratch typically requires large numbers of images to be more effective in extracting the important features of new and unseen Arabic sign images.				
	- Recognition accuracy was unsatisfactory and needs to be improved.				
	- Limited evaluation criteria. Only accuracy and confusion matrices were used.				
	- Insufficient discussion and analysis of the results.				



Table 17. Cont.

[Ref.] (Year)	Model Advantages and Disadvantages	Sign Letters or Words	Accuracy	Is Efficiency Measured?	Is it Bidirectional?
[63] (2021)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Proposed a deep learning model based on a CNN for interpreting Arabic sign letters.</li> <li>- Optimized the model training by using a cross-entropy function with Adam version to minimize the training loss.</li> <li>- Max pooling layers were used in the built model to decrease the number of parameters to minimize over-fitting.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Complex and computationally expensive.</li> <li>- The use of SMOTE on the whole dataset before splitting it into training and test sets does not reflect a real evaluation of the model.</li> <li>- Applying the SMOTE may create overlap and more noisy instances than the original one, which leads to poor performance on unseen test images.</li> <li>- Limited evaluation criteria.</li> <li>- No image enhancement model was used.</li> </ul>	Letters	96.59% (without SMOTE), and 97.29% (with SMOTE)	Not Measured	No
[64] (2022)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Proposed an approach for Arabic sign alphabet recognition using a pre-trained EfficientNetB4 model.</li> <li>- Developed data augmentation with a preprocessing technique on the training set to improve performance.</li> <li>- Introduced a comparative analysis of performance for different pre-trained models.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- EfficientNets works poorly on hardware accelerators.</li> <li>- The performance of the model still needs to be improved.</li> <li>- The model needs an optimization method to select the best parameter values.</li> </ul>	Letters	95%	Not Measured	No
[65] (2022)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Generated video-based Arabic Sign Language datasets.</li> <li>- Proposed transfer learning models to extract video features.</li> <li>- Eliminated useless frames.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Augmenting the whole dataset before splitting it into training and test sets does not reflect a real evaluation of the model.</li> <li>- Combining transfer learning models with RNNs increases the complexity of the model, making it more challenging to train and optimize.</li> <li>- The model is computationally expensive. It requires significant resources for training and recognition.</li> </ul>	Words	93.4%	Not Measured	No
[66] (2022)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Extracted sign image features based on transfer learning models.</li> <li>- Use of data preprocessing and segmentation.</li> <li>- Achieved a competitive recognition result.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- No image enhancement model was used.</li> <li>- Limited evaluation criteria. Only an accuracy metric was used.</li> <li>- Insufficient discussion and analysis of the results.</li> </ul>	Letters	97%	Not Measured	No

Table 17. Cont.

[Ref.] (Year)	Model Advantages and Disadvantages	Sign Letters or Words	Accuracy	Is Efficiency Measured?	Is it Bidirectional?
[33] (2022)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Customized several Arabic sign language datasets.</li> <li>- Created a dataset with natural circumstances and environments.</li> <li>- Use of data preprocessing to reduce the cost time.</li> <li>- Use of two CNN models with an RNN model to improve video feature extraction.</li> <li>- Interpretation not translation.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Using video frames as input led to increase in overall processing time.</li> <li>- Using two CNN models and one RNN model with five layers led to overfitting and low efficiency.</li> <li>- Confusion matrix used only for evaluation measurement.</li> </ul>	Letters and Words	98%	Not measured	No
[67] (2023)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Extracted image features of sign letters using a transfer learning-based EfficientNetB1 model.</li> <li>- Performed data preprocessing (Normalization, Scaling, . . etc.).</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Evaluation of the model was based only on validation accuracy, not testing accuracy.</li> <li>- Limited evaluation criteria. Only an accuracy metric was used.</li> <li>- Insufficient discussion and analysis of the results.</li> </ul>	Letters	97.9% validation accuracy	Not Measured	No
[68] (2023)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Created a new database of 5000 RGB sign letter images.</li> <li>- Combined the created dataset with the existing Arabic sign letters dataset.</li> <li>- Applied a preprocessing stage to enhance the representation of gesture images.</li> <li>- Extracted points' marks from the depth data of hand images.</li> <li>- A deep CNN model was proposed for recognizing the sign letters.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Training the CNN model from scratch requires large amounts of images to be more effective when recognizing new and unseen sign images.</li> <li>- Extracting points' marks and training the CNN model requires significant resources for processing and recognition.</li> <li>- Noisy images and orientation changes may affect the performance of the model.</li> </ul>	Letters	97.07%	Not Measured	No
[34] (2023)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Used different datasets with huge sample sizes.</li> <li>- Used hand edge detection to improve the accuracy.</li> <li>- Used 12 CNN deep learning models for classifications with predication vote.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- No image enhancement model was used.</li> <li>- Constrained environment.</li> <li>- Low efficiency due to video frame usage with 14 images and complex model architecture.</li> <li>- Large depth causes over-fitting.</li> </ul>	Letters	93.7%	Not measured	No

Table 17. Cont.

[Ref.] (Year)	Model Advantages and Disadvantages	Sign Letters or Words	Accuracy	Is Efficiency Measured?	Is it Bidirectional?
[35] (2023)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Used segmentation methods to improve detection.</li> <li>- Simple CNN model architecture was used.</li> <li>- Preprocessing data was used to improve efficiency.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Used a limited dataset with only 14 letters.</li> <li>- Slow training time due to the number of epochs used.</li> <li>- Low efficiency due to imbalanced data.</li> <li>- Poor precision, recall, and F1-score evaluation.</li> </ul>	Letters	97%	measured	No
[36] (2023)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Used real-time EMG signals for data capturing and processing</li> <li>- Efficient performance.</li> <li>- Used a simple model.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Limited dataset with environment constraints.</li> <li>- Experimental evaluation was not performed.</li> <li>- Prone to numerous interferences, such as movement, temperature, electromagnetic radiation, and noise.</li> </ul>	Letters and Numbers	Not measured	Not measured	No
[37] (2024)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Built the largest Arabic Saudi Sign Language (SSL) database.</li> <li>- Established a dataset framework.</li> <li>- Introduced a convolutional graph neural network (CGCN) architecture made up of a small number of 3DGCN layers.</li> <li>- Static and dynamic signs were used.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- No data preprocessing was used.</li> <li>- Limited consideration of the environment.</li> <li>- Experimental evaluation was not performed.</li> <li>- Focused more on dataset creation.</li> </ul>	Words	97%	Not measured	No
[38] (2024)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- A data preparation and preprocessing model was used.</li> <li>- A simple CNN model was utilized.</li> <li>- A dropout strategy was applied to reduce processing time.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- The dataset was small.</li> <li>- Prone to under-fitting issues.</li> <li>- Consumed a large amount of resources due to model implementation steps.</li> <li>- Limited evaluation metrics were used.</li> </ul>	Letters	97%	Not measured	No
[This Work] (2024)	<p>Pros:</p> <ul style="list-style-type: none"> <li>- Created a sufficient dataset of Arabic sign words.</li> <li>- Built a lightweight DNN model for classification.</li> <li>- Optimized the hyper-parameters of a proposed classification model for improving performance.</li> <li>- Use of image enhancement and preprocessing methods.</li> <li>- Compared eight pre-trained CNN models for feature extraction.</li> <li>- Achieved a high performance result compared with the current state-of-the-art models.</li> <li>- It is a bidirectional Arabic sign translation framework.</li> <li>- It is efficient when used for real-time applications.</li> </ul> <p>Cons:</p> <ul style="list-style-type: none"> <li>- Facial expression can be used for supporting sign translation process.</li> <li>- Extra examples with new labels can be added to the dataset to cover more words from the Arabic dictionary.</li> </ul>	Words	98.65%	Measured	Yes

Table 17 shows that the advantages and performance of our proposed work are considered highly suitable for recognizing and translating Arabic sign language images into meaningful text words and vice versa. The novelty of adapting the fuzzy matching score method proves its ability to translate the input text into the appropriate sign image. It can improve its ability to catch and rectify mistakes in the pronunciation of Arabic words, spelling errors, and typos, converting them to the appropriate Arabic sign language images and producing more accurate results.

## 6. Conclusions and Future Work

The study aimed to develop a bidirectional Arabic sign language translator framework. A prototype was implemented based on the proposed framework modules and components using image processing methods with deep learning and fuzzy logic techniques. The automatic bidirectional prototype translator for Arabic sign language was developed to classify 14 classes of hand signs into Arabic text meanings and vice versa. The ArSL dataset was collected carefully based on the specific criteria and consisted of 7030 images for 14 sign classes. The dataset was prepared well, which can be considered a promising Arabic sign language dataset for future research. This study found that preparing an appropriate dataset is essential to solving the domain problem, specifically when used with deep learning and transfer learning applications.

This research can be used as a preliminary study to develop a bidirectional translation system for Arabic Sign language, since there is limited existing research on bidirectional Arabic sign translation. Experimental results showed that the best pre-trained CNN-based Custom DNN model can effectively classify the Arabic sign images with an approximate accuracy of 99%. The performance and efficiency for the ArSL prototype classification task were assessed using a test set and an 80:20 splitting procedure, obtaining accuracy results from the highest to the lowest rates with average classification time in seconds for each utilized model, including (VGG16, 98.65%, 72.5), (MobileNetV2, 98.51%, 100.19), (VGG19, 98.22%, 77.16), (DenseNet121, 98.15%, 80.44), (Xception, 96.44%, 72.54), (NASNetLarge, 96.23%, 84.96), (InceptionV3, 94.31%, 76.98), and (ResNet152, 47.23%, 98.51). Unfortunately, the ResNet152 model failed to classify the test set, with an accuracy rate of 47%. Thus, we suggest excluding it from application in such a problem domain or re-evaluating it using different hyper-parameters.

The overall prototype performance results showed that the study method effectively reduced the processing time of feature extraction and extracted the significant features of signs and images. The proposed prototype proves the ability of selected CNN models to successfully translate the 14 hand-sign labels with the proposed methods with excellent performance. The experimental results showed that the CNN model achieved outstanding efficiency in computation time during the training and testing process. Additionally, the proposed prototype's efficiency results show the ability to develop such a system in mobile devices, specifically with VGG16 or Xception models, because they had the lowest computation time. The fuzzy string matching score's evaluation is mathematically validated by computing the distance between the input word and associative dictionary words.

Furthermore, this study is distinguished from previous studies in supporting bidirectional communication between deaf and ordinary people and improving the accuracy of Arabic word sign classification, where there are limited studies in this area. The study proves the suitability of the new concepts applied here to translate the input text into Arabic sign language using the fuzzy matching score module. The fuzzy matching score approach also reduces the translation time by ignoring spelling errors. It can also be applied to solve the problem of various accents, dialects, synonyms, or words with similar meanings with the same sign image. The new contribution here of applying fuzzy matching scores is not limited to only the Arabic language, but can be customized to work in any other language. The study showed the feasibility of developing a real-time bidirectional automated translation system between deaf and ordinary people.

This study addresses the main tasks of designing a bidirectional translation system between Arabic sign language and text. It provides insight into the ability to develop a fully automatic system for all Arabic signs if the dataset is constructed well. We are working to extend the dataset by including more words and applying different transfer learning strategies, such as fine-tuning and freezing different layers of pre-trained models rather than just using them as feature extractors.

This study has limitations, including the limited size of dataset classes and only 14 signs, while extra common Arabic signs can be selected from the Arabic sign dictionary, where the accuracy and performance of the proposed framework may be affected by extra dataset classes. Furthermore, the performance of transfer learning models can be considered a limitation for real-time application development. Finally, misclassification could occur for some sign images in different environments of dataset criteria or in terms of recognizing some signs that included facial expressions.

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

**Funding:** This research received no external funding.

**Data Availability Statement:** The data that support the findings of this study are available on request from the first author.

**Acknowledgments:** The authors extend their appreciation to the King Salman Center For Disability Research for funding this work through Research Group no KSRG-2023-105.

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

## References

- Hsu, P.-H. Readability of hearing related internet information in traditional Chinese. *Speech Lang. Hear.* **2017**, *23*, 158–166. [[CrossRef](#)]
- Al-Khalifa, H.S. Introducing Arabic sign language for mobile phones. In Proceedings of the International Conference on Computers for Handicapped Persons, Milan, Italy, 11–15 July 2020; pp. 213–220.
- Zahra, A.; Hassan, S.-U.-N.; Hassan, M.S.; Parveen, N.; Park, J.-H.; Iqbal, N.; Khatoun, F.; Atteya, M.R. Effect of physical activity and sedentary sitting time on psychological quality of life of people with and without disabilities: A survey from Saudi Arabia. *Front. Public Health* **2022**, *10*, 998890. [[CrossRef](#)] [[PubMed](#)]
- El-Sadany, T.A.; Hashish, M.A. An Arabic morphological system. *IBM Syst. J.* **1989**, *28*, 600–612. [[CrossRef](#)]
- El-Gayyar, M.; Ibrahim, A.; Sallam, A. The ArSL keyboard for android. In Proceedings of the 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, Egypt, 12–14 December 2015; pp. 481–486.
- Abdel-Fattah, M.A. Arabic sign language: A perspective. *J. Deaf Stud. Deaf Educ.* **2005**, *10*, 212–221. [[CrossRef](#)] [[PubMed](#)]
- Khan, R.U.; Khattak, H.; Wong, W.S.; AlSalman, H.; Mosleh, M.A.; Rahman, M.; Md, S. Intelligent Malaysian Sign Language Translation System Using Convolutional-Based Attention Module with Residual Network. *Comput. Intell. Neurosci.* **2021**, *2021*, 9023010. [[CrossRef](#)] [[PubMed](#)]
- Mohameed, R.A.; Naji, R.M.; Ahmeed, A.M.; Saeed, D.A.; Mosleh, M.A. Automated translation for Yemeni’s Sign Language to Text Using Transfer Learning-based Convolutional Neural Networks. In Proceedings of the 2021 1st International Conference on Emerging Smart Technologies and Applications (eSmarTA), Sana’a, Yemen, 10–12 August 2021; pp. 1–5.
- Damian, S. Spoken vs. Sign Languages—What’s the Difference? *Cogn. Brain Behav.* **2011**, *15*, 251.
- Aronoff, M.; Meir, I.; Sandler, W. The paradox of sign language morphology. *Language* **2005**, *81*, 301. [[CrossRef](#)] [[PubMed](#)]
- Arivazhagan, N.; Bapna, A.; Firat, O.; Lepikhin, D.; Johnson, M.; Krikun, M.; Chen, M.X.; Cao, Y.; Foster, G.; Cherry, C. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv* **2019**, arXiv:1907.05019.
- Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv* **2016**, arXiv:1609.08144.

13. Bao, C.; Ji, H.; Quan, Y.; Shen, Z. Dictionary learning for sparse coding: Algorithms and convergence analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 1356–1369. [[CrossRef](#)]
14. Tomasi, C. Histograms of oriented gradients. *Comput. Vis. Sampl.* **2012**, 1–6. Available online: <https://courses.cs.duke.edu/compsci527/spring19/notes/hog.pdf> (accessed on 15 January 2024).
15. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; pp. 21–37.
16. Nguyen, T.D.; Ranganath, S. Facial expressions in American sign language: Tracking and recognition. *Pattern Recognit.* **2012**, *45*, 1877–1891. [[CrossRef](#)]
17. Amrutha, C.; Davis, N.; Samrutha, K.; Shilpa, N.; Chunkath, J. Improving language acquisition in sensory deficit individuals with mobile application. *Procedia Technol.* **2016**, *24*, 1068–1073. [[CrossRef](#)]
18. Rajam, P.S.; Balakrishnan, G. Real time Indian sign language recognition system to aid deaf-dumb people. In Proceedings of the 2011 IEEE 13th International Conference on Communication Technology (ICCT), Jinan, China, 25–28 September 2011; pp. 737–742.
19. Bhuyan, M.K.; Ramaraju, V.V.; Iwahori, Y. Hand gesture recognition and animation for local hand motions. *Int. J. Mach. Learn. Cybern.* **2014**, *5*, 607–623. [[CrossRef](#)]
20. Gandhi, P.; Dalvi, D.; Gaikwad, P.; Khode, S. Image based sign language recognition on android. *Int. J. Eng. Tech.* **2015**, *1*, 55–60.
21. Lahoti, S.; Kayal, S.; Kumbhare, S.; Suradkar, I.; Pawar, V. Android based american sign language recognition system with skin segmentation and SVM. In Proceedings of the 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bengaluru, India, 10–12 July 2018; pp. 1–6.
22. Ozcan, T.; Basturk, A. Transfer learning-based convolutional neural networks with heuristic optimization for hand gesture recognition. *Neural Comput. Appl.* **2019**, *31*, 8955–8970. [[CrossRef](#)]
23. Aloysius, N.; Geetha, M. Understanding vision-based continuous sign language recognition. *Multimed. Tools Appl.* **2020**, *79*, 22177–22209. [[CrossRef](#)]
24. Imran, J.; Raman, B. Deep motion templates and extreme learning machine for sign language recognition. *Vis. Comput.* **2020**, *36*, 1233–1246. [[CrossRef](#)]
25. Assaleh, K.; Al-Rousan, M. Recognition of Arabic sign language alphabet using polynomial classifiers. *EURASIP J. Adv. Signal Process.* **2005**, *2005*, 507614. [[CrossRef](#)]
26. El-Bendary, N.; Zawbaa, H.M.; Daoud, M.S.; Hassanien, A.E.; Nakamatsu, K. Arslat: Arabic sign language alphabets translator. In Proceedings of the 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), Krakow, Poland, 8–10 October 2010; pp. 590–595.
27. Samir, A.; Tolba, M.F. A Proposed Standardization for Arabic Sign Language Benchmark Database. *Egypt. J. Lang. Eng.* **2015**, *2*, 1–9. [[CrossRef](#)]
28. Ahmed, A.M.; Alez, R.A.; Taha, M.; Tharwat, G. Automatic translation of Arabic sign to Arabic text (ATASAT) system. *J. Comput. Sci. Inf. Technol.* **2016**, *6*, 109–122.
29. Ahmed, A.M.; Alez, R.A.; Tharwat, G.; Taha, M.; Ghribi, W.; Badawy, A.S.; Changalasetty, S.B.; Bose, J.S.C. Towards the design of automatic translation system from Arabic Sign Language to Arabic text. In Proceedings of the 2017 International Conference on Inventive Computing and Informatics (ICICI), Coimbatore, India, 23–24 November 2017; pp. 325–330.
30. Luqman, H.; Mahmoud, S.A. Automatic translation of Arabic text-to-Arabic sign language. *Univers. Access Inf. Soc.* **2019**, *18*, 939–951. [[CrossRef](#)]
31. Aly, S.; Aly, W. DeepArSLR: A novel signer-independent deep learning framework for isolated arabic sign language gestures recognition. *IEEE Access* **2020**, *8*, 83199–83212. [[CrossRef](#)]
32. Kamruzzaman, M. Arabic sign language recognition and generating Arabic speech using convolutional neural network. *Wirel. Commun. Mob. Comput.* **2020**, *2020*, 3685614. [[CrossRef](#)]
33. Balaha, M.M.; El-Kady, S.; Balaha, H.M.; Salama, M.; Emad, E.; Hassan, M.; Saafan, M.M. A vision-based deep learning approach for independent-users Arabic sign language interpretation. *Multimed. Tools Appl.* **2023**, *82*, 6807–6826. [[CrossRef](#)]
34. Nahar, K.M.; Almomani, A.; Shatnawi, N.; Alauthman, M. A robust model for translating arabic sign language into spoken arabic using deep learning. *Intell Autom Soft Comput* **2023**, *37*, 2037–2057. [[CrossRef](#)]
35. AbdElghfar, H.A.; Ahmed, A.M.; Alani, A.A.; AbdElaal, H.M.; Bouallegue, B.; Khattab, M.M.; Tharwat, G.; Youness, H.A. A model for qur’anic sign language recognition based on deep learning algorithms. *J. Sens.* **2023**, *2023*, 9926245. [[CrossRef](#)]
36. Amor, A.B.H.; El Ghoul, O.; Jemni, M. An EMG dataset for Arabic sign language alphabet letters and numbers. *Data Brief* **2023**, *51*, 109770. [[CrossRef](#)]
37. Alsulaiman, M.; Faisal, M.; Mekhtiche, M.; Bencherif, M.; Alrayes, T.; Muhammad, G.; Mathkour, H.; Abdul, W.; Alohal, Y.; Alqahtani, M. Facilitating the communication with deaf people: Building a largest Saudi sign language dataset. *J. King Saud Univ. Comput. Inf. Sci.* **2023**, *35*, 101642. [[CrossRef](#)]
38. El Kharoua, R.; Jiang, X. Deep Learning Recognition for Arabic Alphabet Sign Language RGB Dataset. *J. Comput. Commun.* **2024**, *12*, 32–51. [[CrossRef](#)]
39. Lozano-Diez, A.; Zazo, R.; Toledano, D.T.; Gonzalez-Rodriguez, J. An analysis of the influence of deep neural network (DNN) topology in bottleneck feature based language recognition. *PLoS ONE* **2017**, *12*, e0182580. [[CrossRef](#)] [[PubMed](#)]
40. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [[CrossRef](#)]



41. Goel, A.; Goel, A.K.; Kumar, A. The role of artificial neural network and machine learning in utilizing spatial information. *Spat. Inf. Res.* **2023**, *31*, 275–285. [[CrossRef](#)]
42. Kutyniok, G. *An Introduction to the Mathematics of Deep Learning*; EMS Press: Helsinki, Finland, 2023.
43. Wu, J. *Introduction to Convolutional Neural Networks*; National Key Lab for Novel Software Technology, Nanjing University: Nanjing, China, 2017; Volume 5, p. 495.
44. Mallat, S.; Sciences, E. Understanding deep convolutional networks. *Philos. Trans. R. Soc. A Math. Phys.* **2016**, *374*, 20150203. [[CrossRef](#)]
45. Hussain, M.; Bird, J.J.; Faria, D.R. A study on cnn transfer learning for image classification. In Proceedings of the UK Workshop on Computational Intelligence, Nottingham, UK, 5–7 September 2018; pp. 191–202.
46. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2013**, arXiv:arXiv:13330.
47. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [[CrossRef](#)] [[PubMed](#)]
48. Zadeh, L.A. *Fuzzy Logic, Granular, Fuzzy, and Soft Computing*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 19–49.
49. Rudwan, M.S.M.; Fonou-Dombeu, J.V. Hybridizing Fuzzy String Matching and Machine Learning for Improved Ontology Alignment. *Future Internet* **2023**, *15*, 229. [[CrossRef](#)]
50. Zhang, S.; Hu, Y.; Bian, G. Research on string similarity algorithm based on Levenshtein Distance. In Proceedings of the 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing China, 25–26 March 2017; pp. 2247–2251.
51. Navarro, G. A guided tour to approximate string matching. *ACM Comput. Surv.* **2001**, *33*, 31–88. [[CrossRef](#)]
52. Wang, Z. Automatic and robust hand gesture recognition by SDD features based model matching. *Appl. Intell.* **2022**, *52*, 11288–11299. [[CrossRef](#)]
53. Mosleh, M.A.; Manssor, H.; Malek, S.; Milow, P.; Salleh, A. A preliminary study on automated freshwater algae recognition and classification system. *BMC Bioinform.* **2012**, *13*, S25. [[CrossRef](#)]
54. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **2009**, *22*, 1345–1359. [[CrossRef](#)]
55. Zhuang, F.; Qi, Z.; Duan, K.; Xi, D.; Zhu, Y.; Zhu, H.; Xiong, H.; He, Q. A comprehensive survey on transfer learning. *Proc. IEEE* **2020**, *109*, 43–76. [[CrossRef](#)]
56. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 26 July 2017; pp. 4700–4708.
57. Arslan, B.; Memis, S.; Battinsonmez, E.; Batur, O.Z. Fine-Grained Food Classification Methods on the UEC Food-100 Database. *IEEE Trans. Artif. Intell.* **2021**, *3*, 238–243. [[CrossRef](#)]
58. Serte, S.; Serener, A.; Al-Turjman, F. Deep learning in medical imaging: A brief review. *Trans. Emerg. Telecommun. Technol.* **2020**, *10*, e4080. [[CrossRef](#)]
59. Alsharif, M.; Alsharif, Y.; Yahya, K.; Alomari, O.; Albreem, M.; Jahid, A. Deep learning applications to combat the dissemination of COVID-19 disease: A review. *Eur. Rev. Med. Pharmacol. Sci* **2020**, *24*, 11455–11460. [[PubMed](#)]
60. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
61. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
62. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning transferable architectures for scalable image recognition. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 23 June 2018; pp. 8697–8710.
63. Alani, A.A.; Cosma, G. ArSL-CNN: A convolutional neural network for Arabic sign language gesture recognition. *Indones. J. Electr. Eng. Comput. Sci.* **2021**, *22*, 1096–1107. [[CrossRef](#)]
64. Zakariah, M.; Alotaibi, Y.A.; Koundal, D.; Guo, Y.; Elahi, M.M. Sign language recognition for Arabic alphabets using transfer learning technique. *Comput. Intell. Neurosci.* **2022**, *2022*, 4567989. [[CrossRef](#)] [[PubMed](#)]
65. Mahmoud, E.; Wassif, K.; Bayomi, H. Transfer learning and recurrent neural networks for automatic arabic sign language recognition. In Proceedings of the International Conference on Advanced Machine Learning Technologies and Applications, Cairo, Egypt, 5–7 May 2022; pp. 47–59.
66. Duwairi, R.M.; Halloush, Z.A. Automatic recognition of Arabic alphabets sign language using deep learning. *Int. J. Electr. Comput. Eng.* **2022**, *12*, 2996–3004. [[CrossRef](#)]
67. Dabwan, B.A.; Jadhav, M.E.; Ali, Y.A.; Olayah, F.A. Arabic Sign Language Recognition Using EfficientnetB1 and Transfer Learning Technique. In Proceedings of the 2023 International Conference on IT Innovation and Knowledge Discovery (ITIKD), Manama, Bahrain, 8–9 March 2023; pp. 1–5.
68. Hdioud, B.; Tirari, M.E.H. A Deep Learning based Approach for Recognition of Arabic Sign Language Letters. *Int. J. Adv. Comput. Sci. Appl.* **2023**, *14*. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.