Journal of Computer and Communications, 2024, 12, 32-51
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Deep Learning Recognition for Arabic Alphabet Sign Language RGB Dataset

Rabie El Kharoua, Xiaoming Jiang

School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China
Email: rabie.elkharoua@gmail.com, jxm@ujs.edu.cn

How to cite this paper: El Kharoua, R. and Jiang, X.M. (2024) Deep Learning Recognition for Arabic Alphabet Sign Language RGB Dataset. *Journal of Computer and Communications*, **12**, 32-51.
https://doi.org/10.4236/jcc.2024.123003

Received: February 15, 2024
Accepted: March 8, 2024
Published: March 11, 2024

Abstract

This paper introduces a Convolutional Neural Network (CNN) model for Arabic Sign Language (AASL) recognition, using the AASL dataset. Recognizing the fundamental importance of communication for the hearing-impaired, especially within the Arabic-speaking deaf community, the study emphasizes the critical role of sign language recognition systems. The proposed methodology achieves outstanding accuracy, with the CNN model reaching 99.9% accuracy on the training set and a validation accuracy of 97.4%. This study not only establishes a high-accuracy AASL recognition model but also provides insights into effective dropout strategies. The achieved high accuracy rates position the proposed model as a significant advancement in the field, holding promise for improved communication accessibility for the Arabic-speaking deaf community.

Keywords

Convolutional Neural Network (CNN), AASL Dataset, Dropout, Deep Learning, Communication Technology

1. Introduction

Effective communication is essential for social interaction, and its importance becomes even stronger when considering the obstacles facing the deaf [1] community. For individuals with hearing disabilities, hand gesture recognition emerges as a solid tool capable of bridging communication gaps [2] and offering convenience. Sign language is a structured visual communication form involving hand movements, and it plays a central role in the daily deaf and speech-impaired community interactions [3].

Despite its importance, sign language is not unique nor universally understood, sign language is different from one country to another, posing a substantial

barrier between deaf communities all over the world and the general population. Moreover, the absence of signs for all words necessitates the spelling of certain terms letter by letter [4]. This communication gap necessitates the existence of advanced technologies capable of recognizing and interpreting sign language accurately.

This research aims to address this need by developing a deep learning-based sign language recognition mode, focusing on the Arabic sign language alphabet using the AASL dataset. Recognizing the complexity of hand gesture images, we follow a deep-learning approach, a computer vision methodology known for its prowess in image classification [5]. The novelty of our study lies in the use of the AASL dataset which was never used before, the unique preprocessing of the dataset, and the dropout technique used in the training phase.

Various approaches employing Convolutional Neural Networks (CNNs) have been proposed in many applications to bridge the communication gaps for individuals with hearing impairments. One study by C. M. Bishop *et al.* presented a deep CNN architecture for hand sign language recognition, leveraging both spatial and temporal information with convolutional and recurrent layers [6]. Their approach achieved promising results on benchmark datasets such as American Sign Language (ASL) recognition.

In another study, Z. Zhang *et al.* proposed a multi-modal fusion CNN model for sign language recognition, combining depth and RGB images to enhance feature representation and classification accuracy [7]. The fusion of multiple modalities provided robustness against environmental variations and improved performance in real-world scenarios.

R. Li *et al.* explored the effectiveness of transfer learning in hand sign language recognition using pre-trained CNN models [8]. By fine-tuning CNN architectures pre-trained on large-scale image datasets, they achieved notable improvements in recognition accuracy, demonstrating the potential of transfer learning for this task.

The work of S. Hochreiter and J. Schmidhuber introduced Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, for sequential hand sign language recognition tasks [9]. Their approach effectively captured temporal dependencies within sign sequences, leading to improved recognition performance compared to traditional CNN-based methods.

Recent advancements in CNN-based hand sign language recognition have demonstrated promising results, with approaches ranging from deep CNN architectures to multi-modal fusion and transfer learning strategies. These studies pave the way for developing robust and accurate systems to facilitate communication for individuals with hearing impairments.

Our paper's innovation lies in the novel approach to data preprocessing and the strategic implementation of dropout techniques. These methods are not only applicable to image classification tasks but can also be extended to enhance general neural network training across various domains. Further elaboration on

these advancements will be provided in subsequent sections.

2. Method

Another version of the AASL [10] dataset is made where the background of all images is deleted, this second version of the AASL dataset is then prepossessed and augmented before being fed to the CNN model, the background removal is presented as a way to reduce the noise learned by the model on the training phase especially that the dataset is not big enough to avoid overfitting.

3. Data Preparation and Processing for CNN

The RGB Arabic Alphabet Sign Language (AASL) dataset [10] comprises 7856 images of 5432 by 3830 pixels. ASSL was collected by more than 200 participants and according to the authors, ASSL is the first publicly available RGB dataset for Arabic Alphabet Sign Language, Figure 1 represents a sample extracted from the AASL dataset.

3.1. Data Cleaning

During the first tests and after a manual inspection of the different classes of the AASL dataset, some misclassified figures were found, Another problem found was some unclear figures where it was hard to decide where those specific figures belonged, these problems were fixed by reclassifying these images and deleting the ones that are not clear.

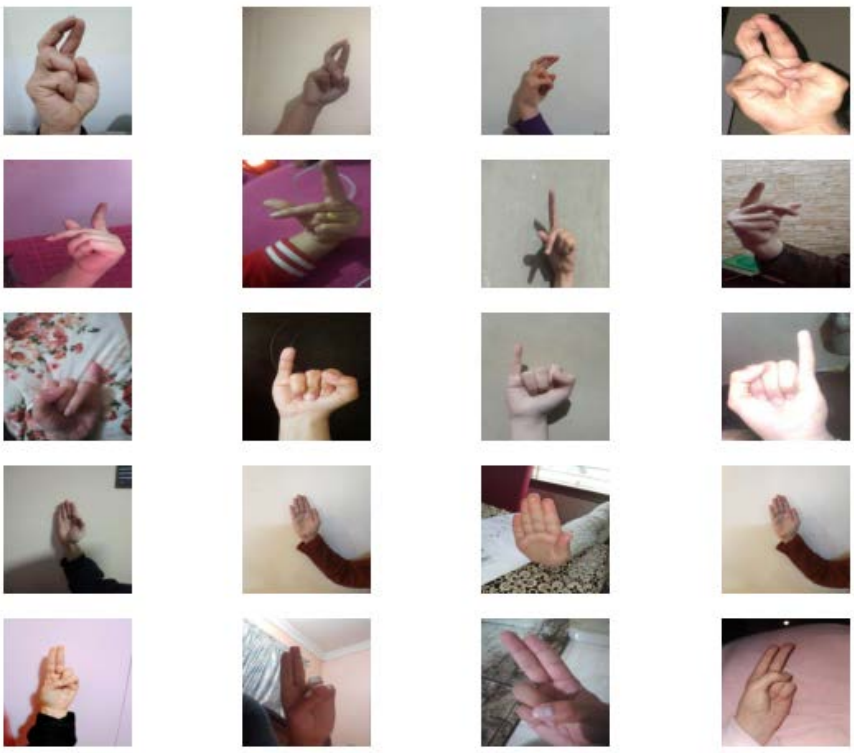

Figure 1. Sample from AASL Dataset.

3.2. Data Resizing

The images of the AASL Dataset [10] have an original size of 5432 by 3830 pixels and were all resized to a size of 224 by 224 pixels.

3.3. AASL Background Removal

To avoid noise and capturing irrelevant features by the CNN model during the training phase, the background of all dataset images was removed, leaving only the hand gesture that the model needs to learn and generalize on. This phase resulted in some corrupted images since background removal is not always easily obtained. Figure 2 displays a sample of images both before and after background removal.

3.4. Final Cleaned Dataset

The Final obtained dataset is a non-background RGB dataset of images with a size of 224 by 224 pixels for each image. The different phases of data preparation resulted in reducing the number of images from 7856 to 6985 images. The distribution shown in Figure 3 represents the distribution of the cleaned version of the AASL dataset.

3.5. Data Split

20% of our dataset is selected randomly to be used as a validation dataset which is 1384 images as seen in Figure 4. The remaining 80% of the data amounts to 5601 samples, as indicated in Figure 5, which will be utilized as the training dataset.

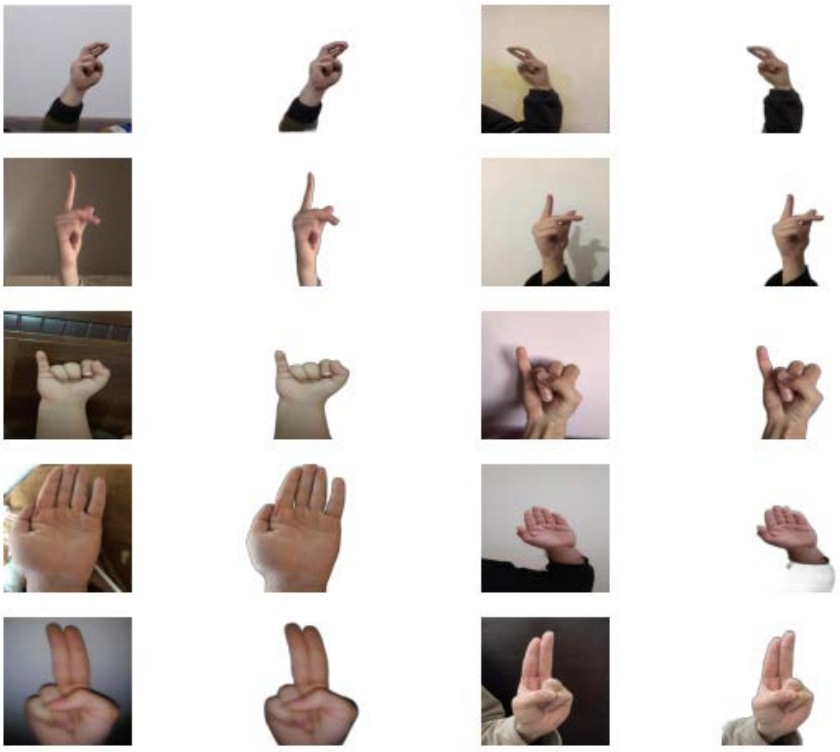

Figure 2. Sample of Images before and after background removal.

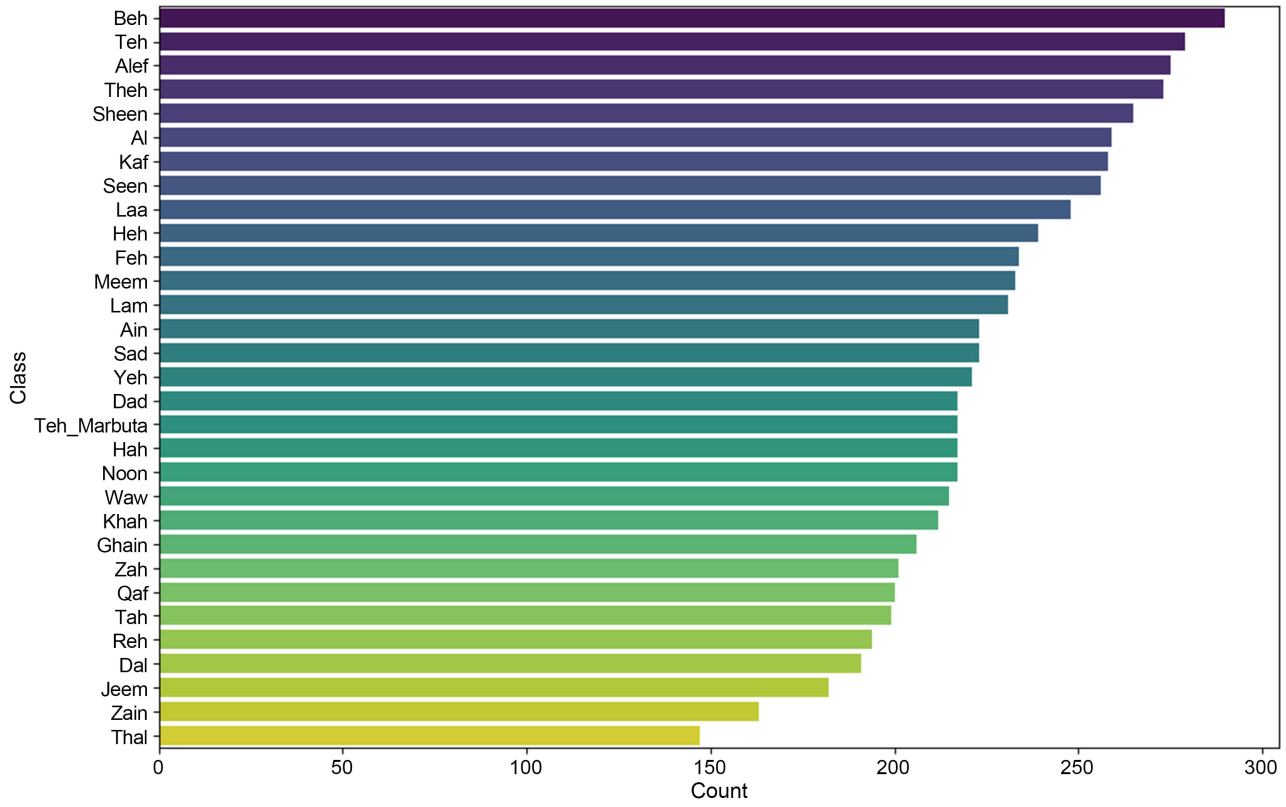

Figure 3. Cleaned AASL dataset distribution.

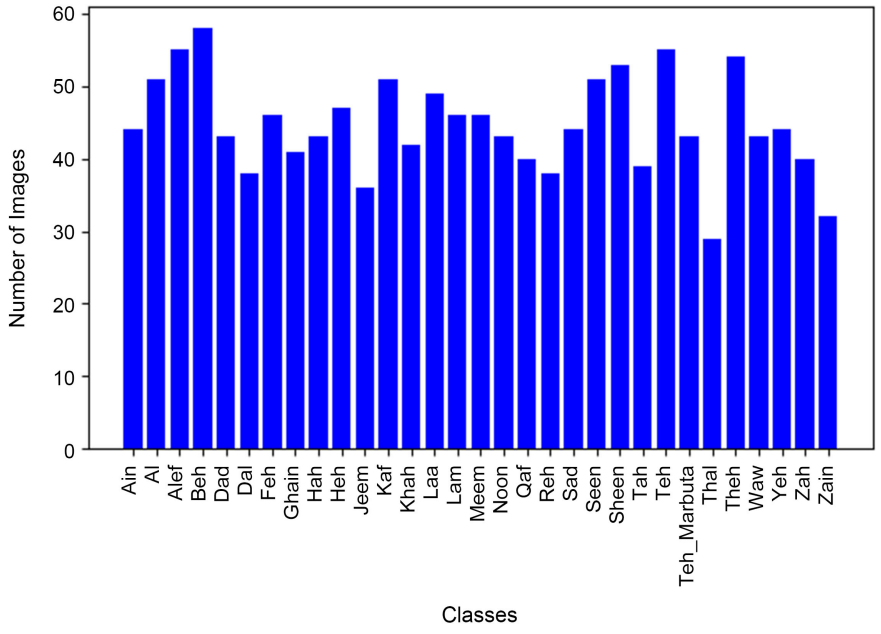

Figure 4. Validation classes distribution.

3.6. Data Augmentation

Since the size of AASL is not big enough and due to the loss of a part of this data during the phase of data cleaning and preparation, data augmentation will be

performed to handle this issue, Figure 6 serves as an illustrative example of this technique. Data augmentation is performed using the "ImageDataGenerator" module from the Keras library [11], the parameters used in our case are the following:

• **Shear Range:** A shear transformation was applied with a range of 30%. This is a deformation effect applied by displacing one part of the image in relation to the other.

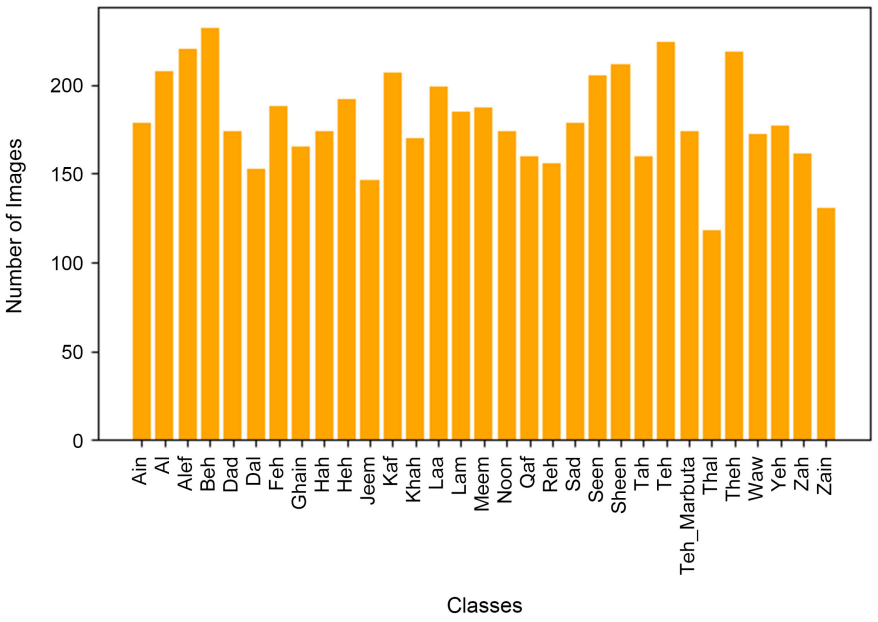

Figure 5. Training classes distribution.

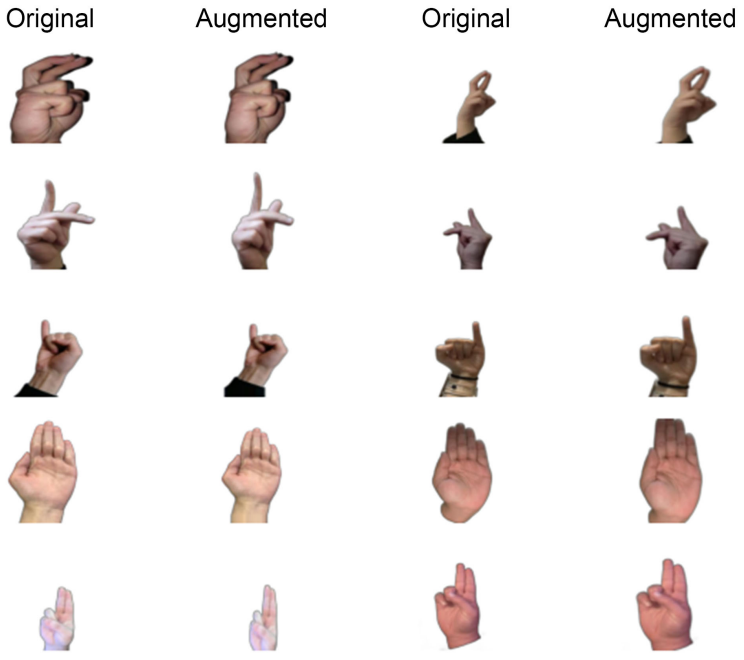

Figure 6. Original and augmented sample from AASL.

• **Zoom Range:** A zoom transformation was applied with a range of 30%. This augmentation magnifies or reduces certain portions of the image.

• **Vertical Flip:** This transformation is a simple vertical flip of the images.

These augmentation strategies help in the creation of a more diverse training dataset which increases the model's capability in generalizing on unseen data [12].

3.7. Convolutional Neural Network

Figure 7 illustrates the Convolutional Neural Network (CNN) utilized, comprising the following essential layers:

• **Convolutional Layers:** Convolutional layers are employed to extract features from the input images, each convolutional layer consists of filters that convolve over the input to capture patterns and create a future map.

• **ReLU Activation:** Rectified Linear Unit (ReLU) is an activation function applied after each convolutional layer, ReLU introduces non-linearity, enabling the model to learn complex relationships in the data [13].

• **MaxPooling Layers:** MaxPooling layers follow the convolutional layers to reduce the spatial dimensions. MaxPooling layers downsample the feature maps while retaining the important information [14].

• **Dropout Layers:** Dropout layers help prevent overfitting by randomly deactivating a fraction of neurons during training, the strategy and rates of dropout layers will be presented in the following sections [15].

• **Flatten Layer:** The flattening layer is employed to transform the feature map from 3D dimensions to a 1D vector.

• **Dense Layers:** The densely connected layers are introduced to capture high-level abstractions from the flattened feature vector.

• **Batch Normalization:** Batch Normalization normalizes the activations of the neurons, ensuring a zero mean and a unit variance. The normalization helps stabilize and expedite the training process by reducing internal covariant shifts [16].

The upcoming sections will outline the configurations employed and the parameters utilized in each layer, including the dimensions of the filters applied in certain layers.

3.8. Model Architecture

Table 1 and Table 2 reveal the Convolutional Neural Network (CNN) utilized, providing a detailed overview of its architecture, including details on each layer with its associated parameters:

The presented CNN model exhibits a total of 22,828,063 parameters where 22,821,919 parameters are trainable, and the rest of the parameters are 6144 non-trainable. The large number of trainable parameters in the model highlights its complexity. This complexity allows it to better understand intricate patterns and details in the input data.

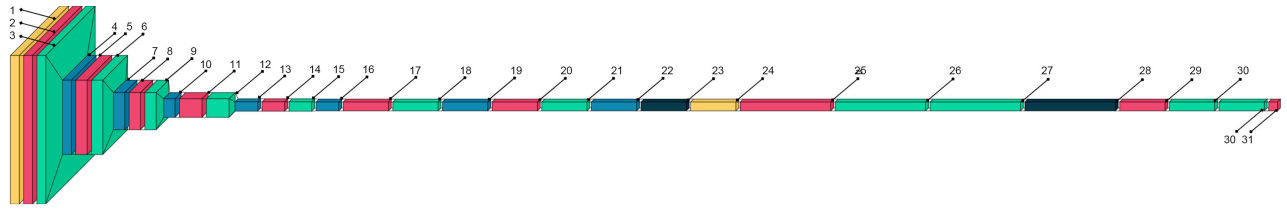

Figure 7. CNN Model visualization.

Table 1. Model architecture.

Layer	Description	Layer Configuration
1	InputLayer	-
2	Conv2D	Filters: 128
3	Activation	-
4	MaxPooling2D	Pooling Window: (2, 2)
5	Conv2D	Filters: 256
6	Activation	ReLu
7	MaxPooling2D	Pooling Window: (2, 2)
8	Conv2D	Filters: 256
9	Activation	ReLu
10	MaxPooling2D	Pooling Window: (2, 2)
11	Conv2D	Filters: 512
12	Activation	ReLu
13	MaxPooling2D	Pooling Window: (2, 2)
14	Conv2D	Filters: 512
15	Activation	ReLu
16	MaxPooling2D	Pooling Window: (2, 2)
17	Conv2D	Filters: 1024
18	Activation	ReLu
19	MaxPooling2D	Pooling Window: (2, 2)
20	Conv2D	Filters: 1024
21	Activation	ReLu
22	MaxPooling2D	Pooling Window: (2, 2)
23	Dropout	Initial Rate: 0.5
24	Flatten	-
25	Dense	Nodes: 2048
26	BatchNormalization	-
27	Activation	-
28	Dropout	Initial Rate: 0.5
29	Dense	Nodes: 1024
30	BatchNormalization	-
31	Activation	ReLu
32	Dense	Nodes: 31

Table 2. Summary of the model parameters.

	Layer Type	Output Shape	Parameters
1	InputLayer	(None, 224, 224, 3)	0
2	Conv2D	(None, 224, 224, 128)	3584
3	Activation	(None, 224, 224, 128)	0
4	MaxPooling2D	(None, 112, 112, 128)	0
5	Conv2D	(None, 112, 112, 256)	295,168
6	Activation	(None, 112, 112, 256)	0
7	MaxPooling2D	(None, 56, 56, 256)	0
8	Conv2D	(None, 56, 56, 256)	590,080
9	Activation	(None, 56, 56, 256)	0
10	MaxPooling2D	(None, 28, 28, 256)	0
11	Conv2D	(None, 28, 28, 512)	1,180,160
12	Activation	(None, 28, 28, 512)	0
13	MaxPooling2D	(None, 14, 14, 512)	0
14	Conv2D	(None, 14, 14, 512)	2,359,808
15	Activation	(None, 14, 14, 512)	0
16	MaxPooling2D	(None, 7, 7, 512)	0
17	Conv2D	(None, 7, 7, 1024)	4,719,616
18	Activation	(None, 7, 7, 1024)	0
19	MaxPooling2D	(None, 3, 3, 1024)	0
20	Conv2D	(None, 3, 3, 1024)	9,438,208
21	Activation	(None, 3, 3, 1024)	0
22	MaxPooling2D	(None, 1, 1, 1024)	0
23	Dropout	(None, 1, 1, 1024)	0
24	Flatten	(None, 1024)	0
25	Dense	(None, 2048)	2,099,200
26	BatchNormalization	(None, 2048)	8192
27	Activation	(None, 2048)	0
28	Dropout	(None, 2048)	0
29	Dense	(None, 1024)	2,098,176
30	BatchNormalization	(None, 1024)	4096
31	Activation	(None, 1024)	0
32	Dense	(None, 31)	31,775

The Figure illustrates the proposed CNN model architecture, providing a visual representation of each layer. The layer numbers in the figure align with the numbers specified in the Table.

4. Model Training and Results

4.1. Experimental Setup

The framework employed in constructing the system was Keras, with TensorFlow serving as the underlying engine. A high-performance workstation was used with 4 CPUs and 2 Nvidia Tesla T4 GPUs [17], more information is presented in Table 3.

4.2. Dropout Strategy

To make the model achieve a high validation accuracy in a short period of training time, a dropout rate change strategy was executed. The model is capable of converging to a high validation accuracy just with a fixed dropout rate but in this case, the training time will be longer and 250 epochs will not be enough to achieve the stated validation accuracy. In many cases even increasing the number of training epochs was not enough to achieve the stated validation accuracy which highlights the importance of this dropout strategy. Figure 8 illustrates the dropout strategy, which comprises the following phases:

Table 3. High-Performance desktop computer specs.

Parameter	Value
CPUs	4
Memory (GiB)	31.36
Memory per CPU (GiB)	7.84
CPU Architecture	64 bit
GPU	2
GPU Architecture	Tesla T4
Video Memory (GiB)	14.75 Per GPU
FPGA	0

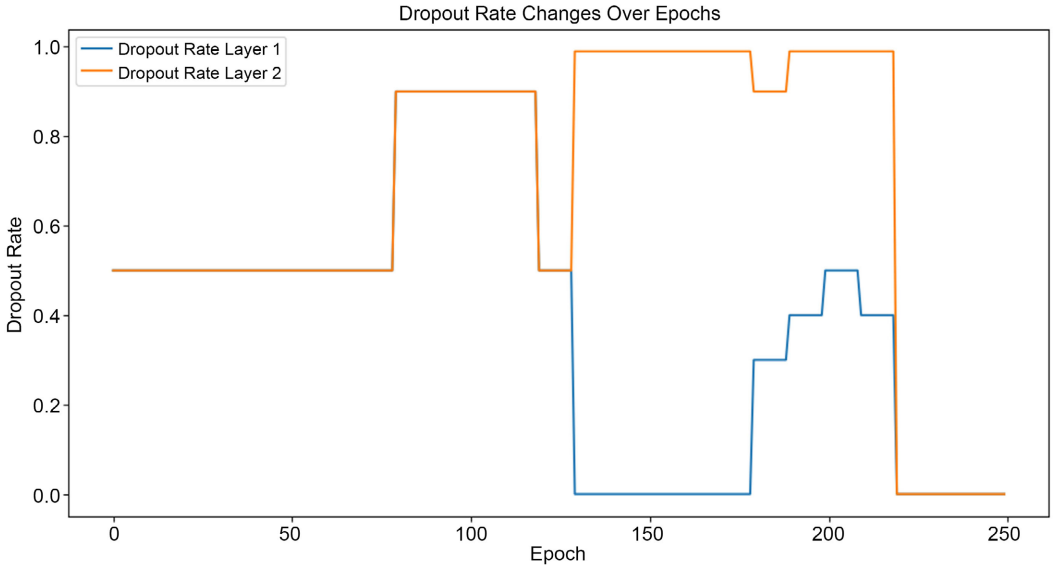

Figure 8. Dropout rate change.

• **Initial Phase:** From epoch 1 to 79 the value of both dropout rates was fixed at 0.5 allowing the model a stable learning phase without disturbance.

• **Suppression Phase:** From epoch 80 to 119, both dropout rates were elevated to 0.9 to challenge the model to classify the images using only 10% of the neurons.

• **Interphase:** From epoch 120 to 129 the values of both dropout rates are set back to the initial rate of 0.5 allowing the model to adjust its weights again and preparing it to the next phase.

• **Extreme Suppression:** From epoch 130 to 219 the first dropout rate was set to 0.0 while the second one was set to 0.99 challenging the model to use only 1% of its neurons to classify the images. During this phase, the first dropout rate was changed over time slowly going up from a value of 0.0 to 0.5.

• **Complete Release:** From epoch 220 to 250 we give a value of 0.0 for both dropout rates allowing the model to use its full capacities to classify the images, this phase is important to fight the underfitting faced during past phases.

The highest validation accuracy achieved was not possible without this dropout strategy which highlights the importance and the role of this strategy.

4.3. Training and Validation Accuracy

The training parameters can be found in Table 4, in this section, we will assess the effectiveness of our CNN model by examining the performance metrics shown in Table 5.

In Figure 9, the depicted chart illustrates the training and validation accuracy trends [18]. Starting with a low value in the initial epochs, the accuracy progressively rises over subsequent epochs. Notably, the validation data attains a peak accuracy of 97.4%, signifying the model's adeptness at generalizing its learned features.

The successive drop in accuracy is due to the extreme change in the dropout rate, which is normal since only a fraction of the neurons will be used, hence the model needs to adjust the few neurons' weights to be capable of classifying all classes using only a small number of neurons; The same drop is seen on training and validation loss.

Table 4. Training parameters.

Parameter	Value
Batch Size	50
Epochs	250

Table 5. Model performance.

Parameter	Value	Comment
Training-data	Accuracy	99.4%
Validation-data	Accuracy	97.4%

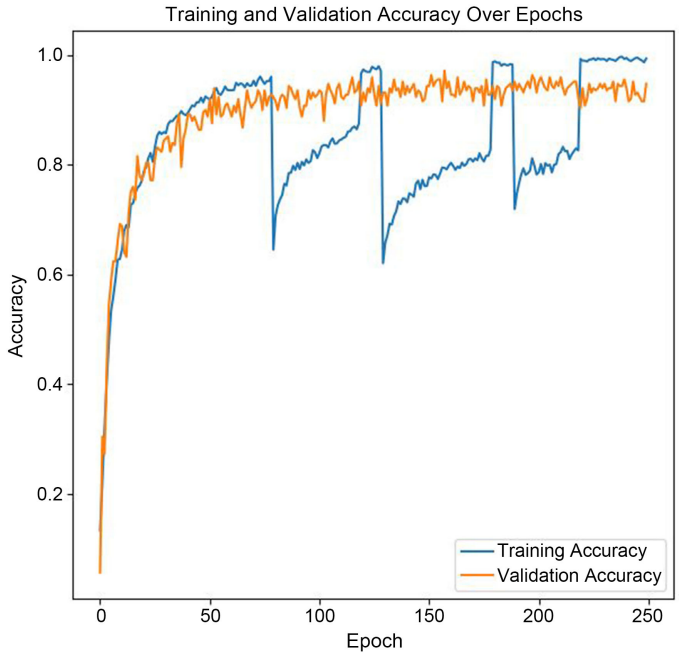

Figure 9. Training and validation accuracy.

The training and validation losses exhibit a descending trajectory, culminating in minimal values for both metrics. This pattern is indicative of a well-fitted model that effectively captures the data without overfitting, as shown in **Figure 10**.

4.4. Precision, Recall and F1-Score

The dataset is imbalanced, which means the accuracy and validation accuracy are not enough to evaluate the performance of our model, for this matter Precision, Recall, and F1-Score [19] are calculated. The weighted average (W.avg) of these metrics is used to evaluate the performance. The CNN model used achieved a performance of 97% across the three metrics. More details are shown in the classification report in **Table 6**.

The confusion matrix [20] is a perfect way to distinguish between true positives, true negatives, false positives, and false negatives which provides good insight into the system's accuracy and its errors. The confusion matrix of the CNN model is shown in **Figure 11** which proves that the performance of the CNN model is perfect with negligible inaccuracies.

4.5. Results Overview

The results underscore the CNN model's exceptional efficacy across all classes. The model consistently achieves precise classification for all categories, except for a few distinctive cases. Within the confusion matrix, the highest value observed is 6, associated with the "Qaf" and "Heh" classes. This confusion can be ascribed to the inherent similarity between these two classes, leading to a degree of overlap.

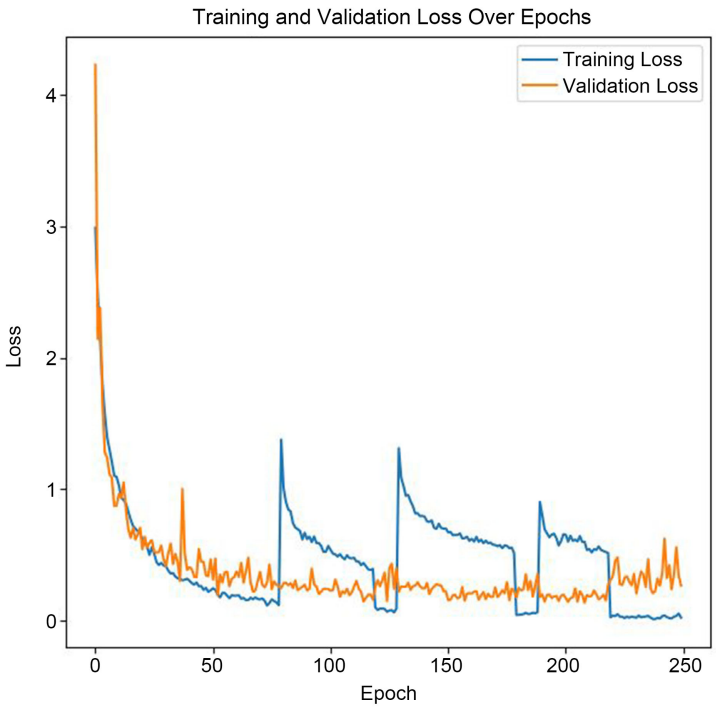

Figure 10. Loss function.

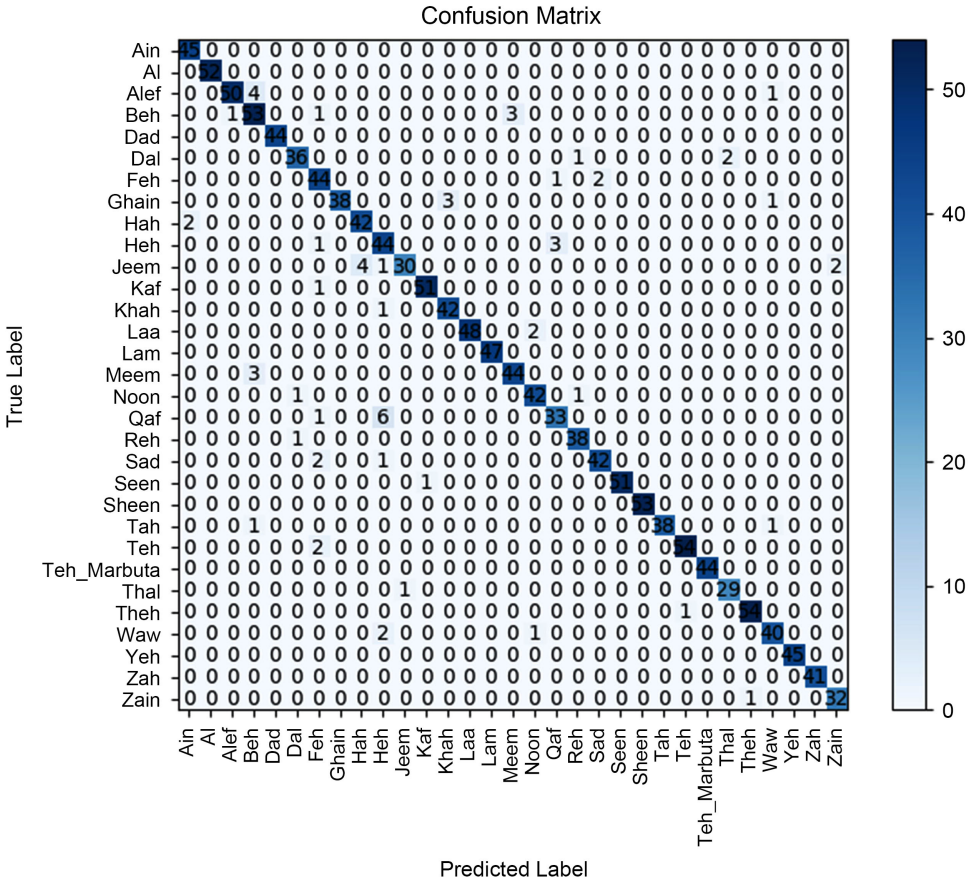

Figure 11. Confusion matrix.

Table 6. Classification report.

Class	Precision	Recall	F1-Score	Support
Ain	0.96	1.00	0.98	45
Al	1.00	1.00	1.00	52
Alef	0.98	0.98	0.98	55
Beh	0.92	0.98	0.95	58
Dad	1.00	0.95	0.98	44
Dal	1.00	0.82	0.90	39
Feh	0.83	0.94	0.88	47
Ghain	1.00	0.95	0.98	42
Hah	0.91	0.95	0.93	44
Heh	0.90	0.92	0.91	48
Jeem	0.94	0.81	0.87	37
Kaf	1.00	0.96	0.98	52
Khah	0.96	1.00	0.98	43
Laa	1.00	1.00	1.00	50
Lam	1.00	1.00	1.00	47
Meem	0.98	0.96	0.97	47
Noon	0.96	1.00	0.98	44
Qaf	0.88	0.88	0.88	40
Reh	0.91	1.00	0.95	39
Sad	0.95	0.89	0.92	45
Seen	1.00	1.00	1.00	52
Sheen	1.00	1.00	1.00	53
Tah	0.95	0.93	0.94	40
Teh	1.00	0.96	0.98	56
Teh_Marbuta	1.00	1.00	1.00	44
Thal	0.90	0.93	0.92	30
Theh	0.96	0.98	0.97	55
Waw	1.00	0.98	0.99	43
Yeh	1.00	1.00	1.00	45
Zah	1.00	0.93	0.96	41
Zain	0.89	1.00	0.94	33
Accuracy			0.96	1410
Macro Avg	0.96	0.96	0.96	1410
Weighted Avg	0.96	0.96	0.96	1410

4.6. Performance Verification

In this phase, we will test our CNN model using a sample from another dataset, the background of the images is removed, we apply the same preprocessing as before and we feed the images to the CNN model. The model shows a perfect performance across all images as seen in Figure 12.

4.7. Model Predictions Explanation

4.7.1. Using Lime

LIME (Local Interpretable Model-agnostic Explanations) [21] is a framework for explaining the predictions for machine learning models by perturbing input features and observing the changes in predictions. Lime is used in our case for

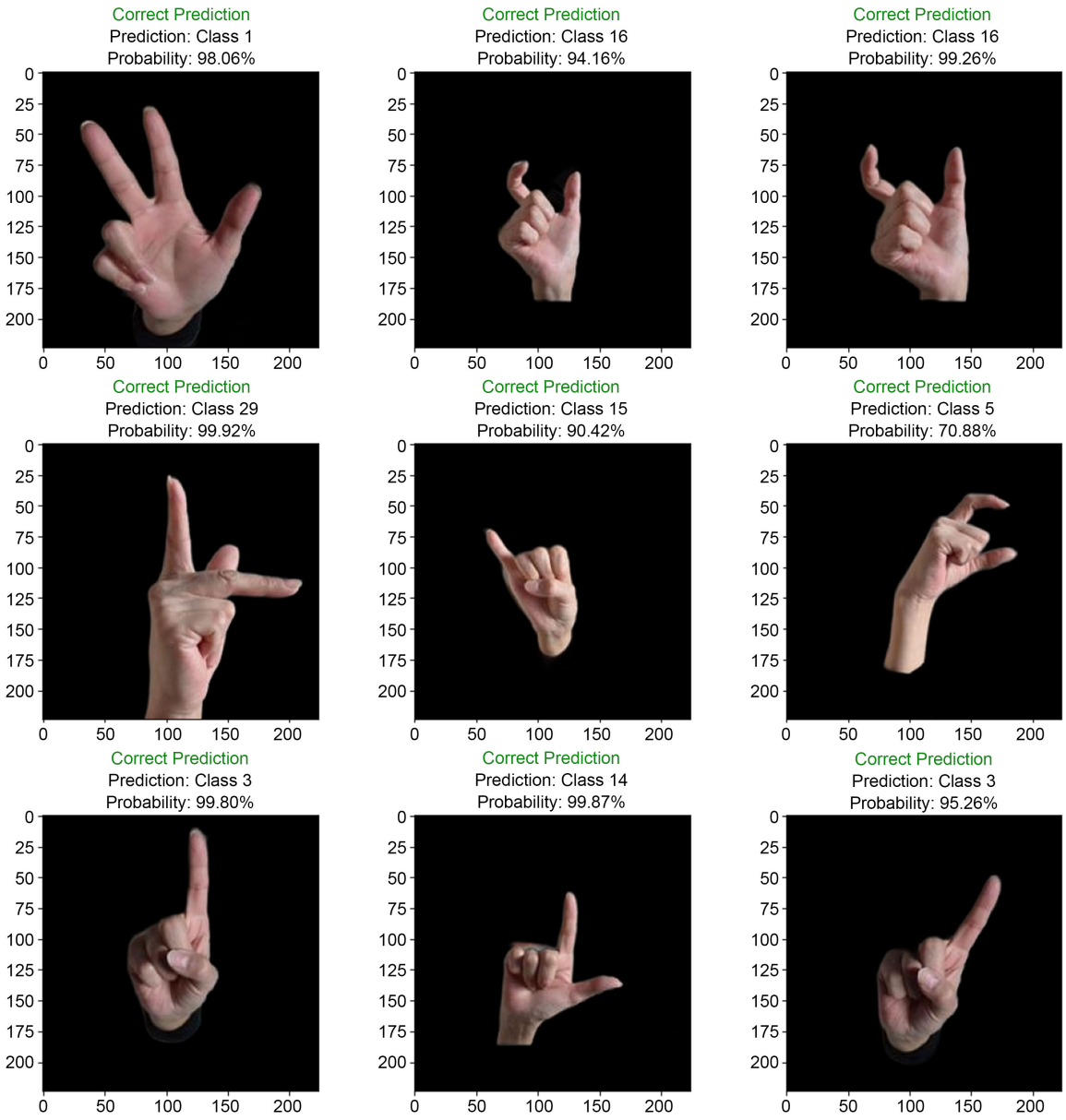

Figure 12. Performance verification.

the same images used in the performance verification phase where regions are visualized using green and red colors, with yellow lines marking the boundaries of the interesting parts. The goal is to make the model's decision understandable and interpretable.

The findings shown in Figure 13 indicate that the classification of hand gestures heavily relies on specific and discernible regions, while other portions of the hand are systematically disregarded. This observation is particularly noteworthy, as certain hand segments are recurrent across multiple classes. Consequently, the model tends to overlook these recurrent segments, deeming them inconsequential in distinguishing between distinct classes. This phenomenon underscores the model's capacity to focus on the salient aspects of hand gestures crucial for accurate classification, thereby enhancing its discriminatory capabilities.

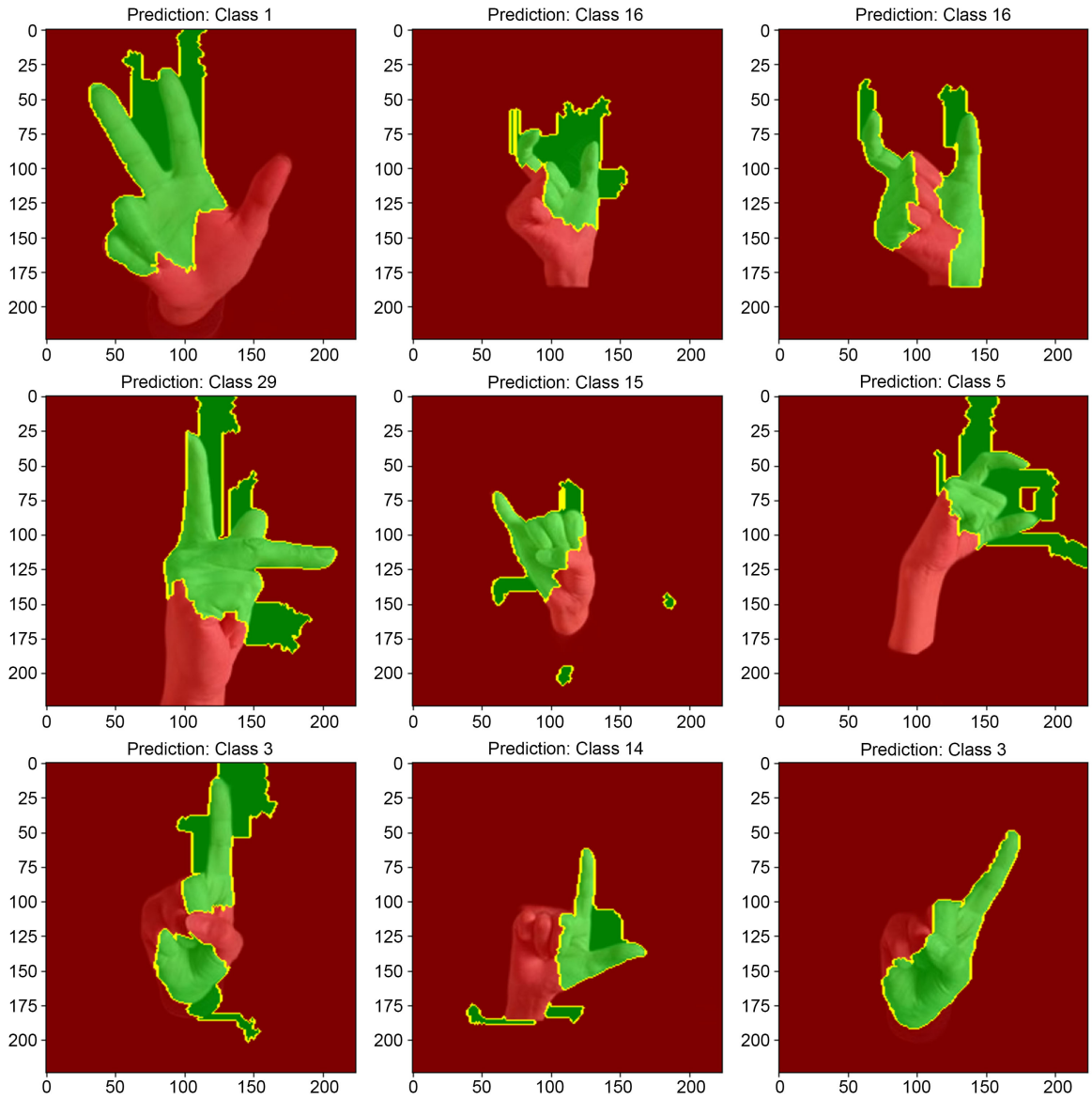

Figure 13. Model predictions explanation with LIME.

4.7.2. Using LASSO

LASSO (Least Absolute Shrinkage and Selection Operator) [22] is a regularization technique used in machine learning and statistics. In the context of linear models, LASSO adds a penalty term to the objective function, encouraging the model to prefer sparse solutions by shrinking some coefficients to exactly zero. This promotes feature selection and can be used to identify the most important features in a model. In our case, LASSO performed almost the same as Lime which confirms the features learned by the model as seen in Figure 14. The red dots represent the most important parts or features learned by the model.

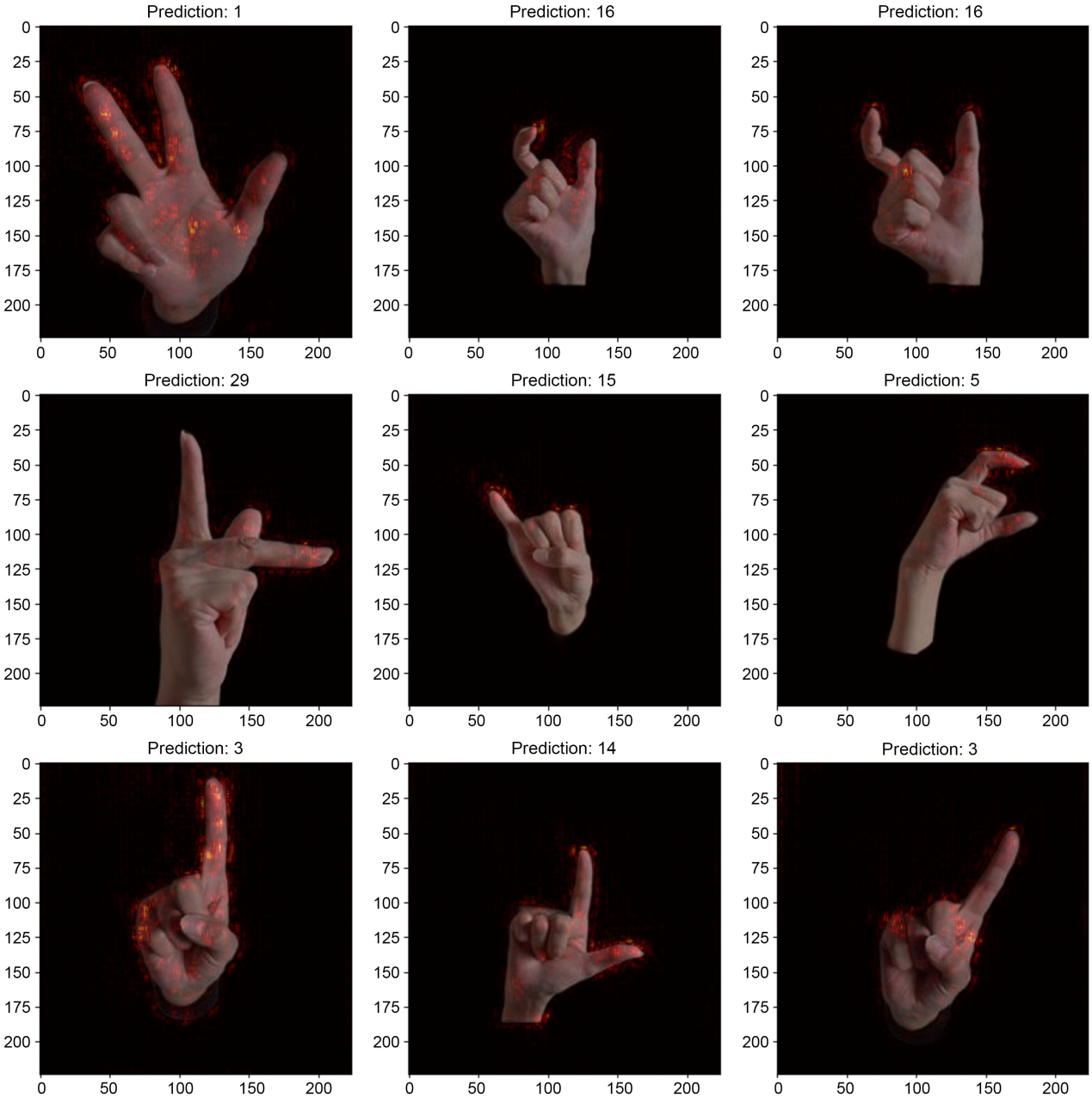

Figure 14. LASSO for verification data.

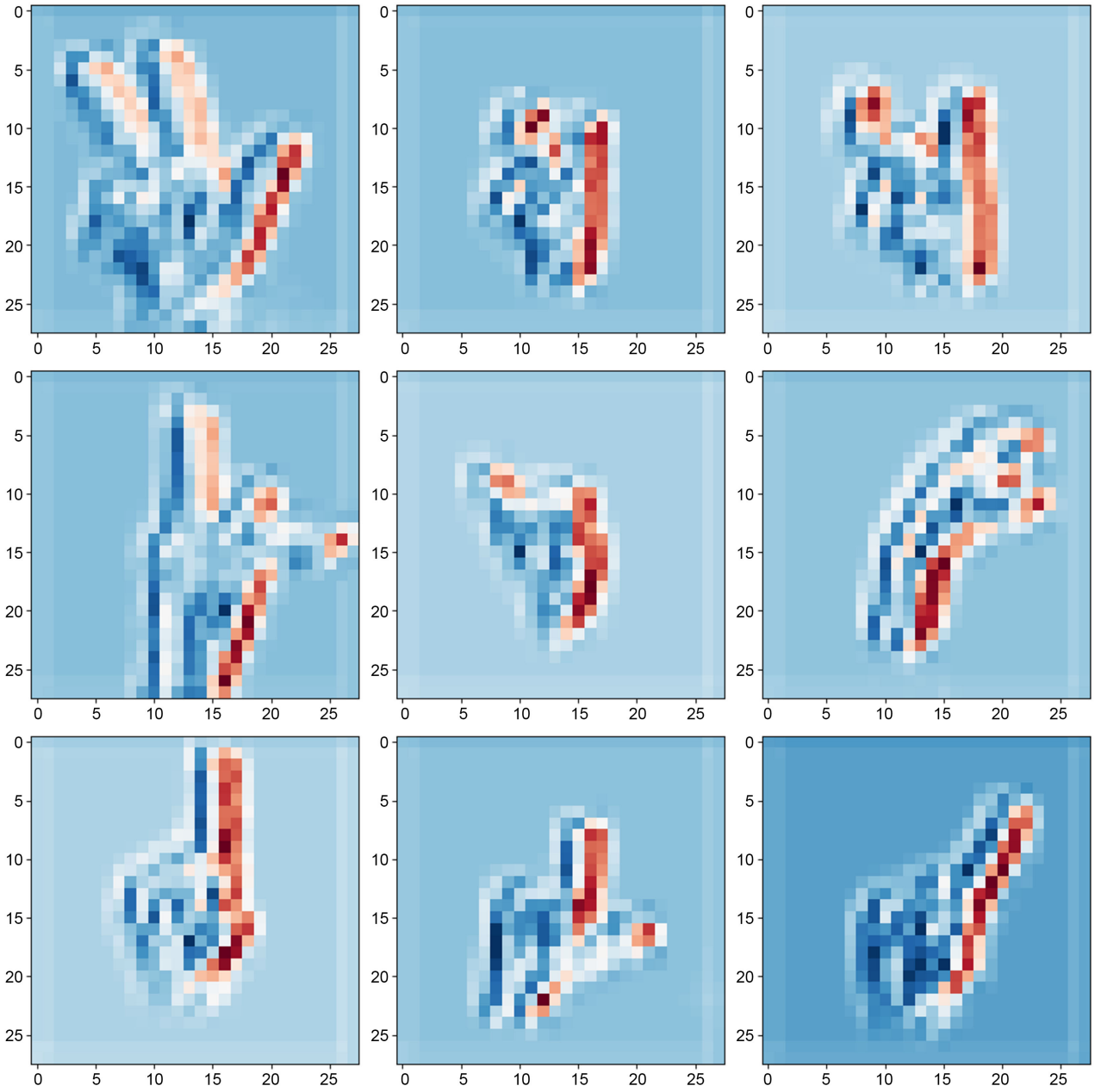

Figure 15. Convolutional filter visualization.

4.7.3. Using Filters of Convolution Layers

We visualize in this section one of the filters of the third convolutional layer applied to the same data, this specific layer is chosen on purpose since it gives good insights into what the model is learning, the more in-depth layers are more complex and not easy to understand. Figure 15 shows the important features detected by this filter in red color where the less important features or pixels are represented in blue color.

5. Conclusions

In this paper, we proposed the first CNN model for Arabic Alphabet Sign Lan-

guage classification using the AASL dataset.

The architecture of this model has been described in detail. The results of the test, validation, and verification steps are described in detail along with the unique dropout strategy employed. The CNN model has shown perfect performance in categorizing the 31 categories of the AASL dataset with an accuracy score of 97.4% and an accuracy of 100% in the verification performance phase.

To the best of our knowledge, this is the first CNN model trained on the AASL dataset. This contribution provided the first step in the development of machine-learning models dedicated to Arabic Sign Language.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Tolentino, L.K.S., Juan, R.O.S., Thio-ac, A.C., Pamahoy, M.A.B., Forteza, J.R.R. and Garcia, X.J.O. (2019). Static Sign Language Recognition Using Deep Learning. *International Journal of Machine Learning and Computing*, **9**, 821-827. https://doi.org/10.18178/ijmlc.2019.9.6.879

[2] Rastgoo, R., Kiani, K. and Escalera, S. (2020) Hand Sign Language Recognition Using Multi-View Hand Skeleton. *Expert Systems with Applications*, **150**, Article ID: 113336. https://doi.org/10.1016/j.eswa.2020.113336

[3] Hossen, M.A., Govindaiah, A., Sultana, S. and Bhuiyan, A. (2018) Bengali Sign Language Recognition Using Deep Convolutional Neural Network. 2018 *Joint 7th International Conference on Informatics, Electronics & Vision* (*ICIEV*) *and* 2018 2*nd International Conference on Imaging, Vision & Pattern Recognition* (*icIVPR*), Kitakyushu, 25-29 June 2018, 369-373. https://doi.org/10.1109/ICIEV.2018.8640962

[4] Chong, T.W. and Lee, B.G. (2018) American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach. *Sensors*, **18**, Article 3554. https://doi.org/10.3390/s18103554

[5] Maier, A., Syben, C., Lasser, T. and Riess, C. (2019) A Gentle Introduction to Deep Learning in Medical Image Processing. *Zeitschrift für Medizinische Physik*, **29**, 86-101. https://doi.org/10.1016/j.zemedi.2018.12.003

[6] Bishop, C.M., *et al.* (2016) Deep Convolutional Networks for Continuous Sign Language Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **38**, 153-162.

[7] Zhang, Z., *et al.* (2018) Multi-Modal Deep Learning for Sign Language Recognition. *IEEE Transactions on Multimedia*, **20**, 1636-1647.

[8] Li, R., *et al.* (2018) Transfer Learning for Sign Language Recognition Using Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, **29**, 5339-5350.

[9] Hochreiter, S. and Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Computation*, **9**, 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

[10] Al-Barham, M., *et al.* (2023) RGB Arabic Alphabets Sign Language Dataset. arXiv: 2301.11932.

[11] Chollet, F., *et al.* (2015) Keras. GitHub Repository.

https://github.com/keras-team/keras

[12] LeCun, Y., *et al.* (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324. https://doi.org/10.1109/5.726791

[13] Glorot, X., *et al.* (2010) Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa Israel, 21-24 June 2010, 807-814.

[14] Scherer, D., *et al.* (2010) Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In: Diamantaras, K., Duch, W. and Iliadis, L.S., Eds., *Artificial Neural Networks—ICANN* 2010, Springer, Berlin, 92-101. https://doi.org/10.1007/978-3-642-15825-4_10

[15] Srivastava, N., *et al.* (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, **15**, 1929-1958.

[16] Ioffe, S. and Szegedy, C. (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv: 1502.03167.

[17] Abadi, M., *et al.* (2016) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. arXiv: 1603.04467.

[18] Sokolova, M. and Lapalme, G. (2009) A Systematic Analysis of Performance mEasures for Classification Tasks. *Information Processing & Management*, **45**, 427-437. https://doi.org/10.1016/j.ipm.2009.03.002

[19] He, H. and Garcia, E.A. (2009) Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, **21**, 1263-1284. https://doi.org/10.1109/TKDE.2008.239

[20] Düntsch, I. and Gediga, G. (2019) Confusion Matrices and Rough Set Data Analysis. *Journal of Physics*: *Conference Series*, **1229**, Article ID: 012055. https://doi.org/10.1088/1742-6596/1229/1/012055

[21] Ribeiro, M.T., Singh, S. and Guestrin, C. (2016) Why should I Trust You?: Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD*), San Francisco, 13-17 August 2016, 1135-1144. https://doi.org/10.1145/2939672.2939778

[22] Tibshirani, R. (1996) Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society*: *Series B* (*Methodological*), **58**, 267-288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x