

Paper Type: Original Article

ArSL-EdgeBench: Duplicate-Controlled and Edge-Oriented Benchmarking for Static Arabic Sign Alphabet Recognition

Khaled Elbehiry¹  and Hussam Elbehiry^{2,*} 

¹ Faculty of Computers Science, Open Institute of Technology (OPIT), Italy, EU.

² Department of Computer Science, Faculty of Information Systems and Computer Science, October 6 University, Giza 12585, Egypt; h.elbehiry.csis@o6u.edu.eg.

Received: 17 Jan 2026

Revised: 01 May 2026

Accepted: 28 Jul 2026

Published: 30 Jul 2026

Abstract

Reported performance in static Arabic sign alphabet recognition is often difficult to compare because duplicate images, variation across random seeds, robustness to image degradation, and deployment cost are rarely examined in the same experiment. In this study, we introduce ArSL-EdgeBench, a reproducible comparison of four image classifiers on the public ArSL2018 dataset. The initial data set comprised 54,049 images belonging to 32 classes. No partition was generated until duplication detection had been performed based on SHA-256, repeated visual hash inside the same class, and label conflict identification via perceptual hashing. The resulting audit retained 48,093 images that were split into equal training, validation, and test fixed stratified sets according to 70:15:15 proportions. Three models – a lightweight separable CNN MobileNetV3-Small, EfficientNetB0, and ResNet50 were trained using three random seeds each time. EfficientNetB0 came first and was chosen solely based on mean validation macro-F1 score. It demonstrated 94.21% $\pm$ 0.21% mean test accuracy, 94.20% $\pm$ 0.19% macro-F1, and 98.84% top-3 accuracy. Seed-42 variant achieved 94.44% accuracy on the clean test set. Its performance remained almost intact under small angle rotation and reduced illumination, while Gaussian noise decreased macro-F1 from 94.42% to 74.46%. Conversion of the model to Float16 TensorFlow Lite allowed reduction of its size to 8.03 MB with 93.14% accuracy retained; the median single-thread CPU latency was 4.12 ms. Overall, these results demonstrate that EfficientNetB0 is an effective accuracy-efficiency baseline for static alphabet prototypes. However, they do not create continuous translation or generalize to novel signers since signer identity information is not available in the public dataset.

Keywords: Arabic Sign Language; ArSL2018; Image de-duplication; Transfer Learning; Robustness; Edge Artificial Intelligence; TensorFlow Lite.

1 | Introduction

Recognition systems for Arabic Sign Language could improve the accessibility of everyday information and services for those people who communicate through signs. Nonetheless, the fundamental challenge is frequently over-simplified to be simple image classification. Sign language includes not only hand shapes but also the movement, facial expressions, body posture and linguistic context. An assembly of images representing an alphabet represents merely one of the facets of that system. This is why the current research is intentionally presented in the form of isolated alphabet recognition rather than sign language translation.



Corresponding Author: h.elbehiry.csis@o6u.edu.eg



<https://doi.org/10.61356/j.scin.2026.3666>



Licensee SciNexuses. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0>).

ArSL2018 dataset has been publicly released as a benchmark dataset consisting of 54,049 annotated images corresponding to 32 Arabic signs and alphabet categories from 40 individuals [1]. Its large amount and wide availability have led to the investigation of conventional machine learning approaches, convolutional neural networks, transfer learning approaches and more recently transformers. However, reviews of Arabic Sign Language research show substantial variation in preprocessing, data splitting, class balancing, model selection, and the way isolated recognition is distinguished from continuous signing [2, 3]. As a result, accuracy figures reported by different studies are not always directly comparable.

The structure of the dataset matters as much as the choice of model. KArSL, for example, provides dynamic multimodal recordings and information about individual signers [4], whereas newer resources increasingly capture facial and body cues in addition to hand movements [5]. ArSL2018 remains valuable for static-image experiments, but the public release does not identify the participants associated with each image. An image-level random split therefore cannot show how well a model will perform on entirely unseen signers. Duplicate or near-repeated files raise a second concern: when closely related samples appear in both the training and test sets, the resulting evaluation may be easier than the intended generalisation problem.

Recent work has achieved impressive results with classical classifiers, pretrained convolutional networks, vision transformers, landmark-based temporal models, and hybrid architectures [6-11]. Those advances are important, but a high accuracy value does not by itself show that a result is reproducible, stable across random initialisations, resistant to routine image degradation, or suitable for deployment on a resource-constrained device. Data leakage is another well-documented source of overly optimistic machine-learning results [12], which makes transparent dataset auditing especially important in this setting.

ArSL-EdgeBench was developed to examine these practical and methodological issues together. We first audited the images for exact and perceptual duplicates before creating any partition. We then used a fixed stratified split with a held-out test set, compared one compact custom CNN with three pretrained architectures across three random seeds, and selected the final architecture using validation macro-F1 alone. The selected model was subsequently examined at class level, challenged with several controlled image corruptions, and exported to Float16 TensorFlow Lite so that accuracy, model size, and inference latency could be considered in the same benchmark. The goal here is not to achieve a new state of the art, but to provide an explicitly reproducible discussion of the trade-offs.

2 | Related Work

Previous research into recognizing characters of the Arabic alphabet used to involve either handcrafted segmentation/feature extraction techniques together with traditional machine learning methods. Thus, Tharwat et al. conducted experiments with supervised learning algorithms using Arabic alphabet images and observed quite good performance of K-Nearest Neighbours [6]. However, nowadays the focus lies on pre-trained backbone visual architectures and transformers. Alharthi and Alzahrani investigated transfer learning models and ViT and Swin variants [7], and Alyami et al. used hand/facial landmarks together with temporal and transformer models [8]. What is quite interesting about their approach is the big difference between signer-dependent and signer-independent scenarios: they test different levels of generalization.

Another related avenue of work concentrated on real-time recognition and deployment of systems. In this direction, Noor et al. have developed a real-time Arabic sign recognition system based on convolutional and recurrent network architectures [9]. Edge information and convolutional and ResNet50 models were used by Ismaiel et al. to achieve 97.14% and 96.88% accuracy on the ArASL dataset, respectively [10]. Recently, CLIP-ArASL has obtained 99.25% $\pm$ 0.14% supervised accuracy by combining image features produced by EfficientNetB0 with bilingual descriptions on a balanced subset of ArSL2018 with 400 images per class [11]. The contributions of these papers differ in terms of techniques, but their performance numbers should not be compared to each other due to the differences in sampling, splitting, augmentation, and task definitions.

Table 1. Methodological context for ArSL-EdgeBench and selected recent Arabic sign recognition studies.

Study	Data and approach	Reported result	Primary emphasis
Tharwat et al. [6]	Arabic alphabet images with preprocessing and conventional classifiers	KNN accuracy of 99.5%	Feature engineering and classifier comparison
Alharthi and Alzahrani [7]	ArSL image classification with transfer learning, ViT, and Swin	High recognition performance	Comparison of visual architectures
Alyami et al. [8]	KArSL-100 landmarks with temporal and transformer models	99.74% signer-dependent; 68.2% signer-independent	Effect of signer-level evaluation
Ismaiel et al. [10]	ArASL and AASL using Canny edges with CNN and ResNet50	97.14% and 96.88% on ArASL	Edge-enhanced visual features
Alasmari [11]	Balanced ArSL2018 and ArSL21L with EfficientNetB0 and bilingual text	99.25% +/- 0.14% on ArSL2018	Supervised and zero-shot multimodal learning
ArSL-EdgeBench	Full duplicate-controlled ArSL2018 with four models, three seeds, corruption tests, and TFLite export	94.21% +/- 0.21% mean test accuracy	Reproducibility, robustness, and edge deployment

Table 1 thus aims to contextualise the studies as opposed to ranking them. In fact, a relatively low score achieved by a more explicit or conservative methodology could provide more insight into generalisability than a relatively high score obtained from a completely different methodology for sample selection or otherwise. This is why ArSL-EdgeBench uses the dataset audit, partition sizes, seed variance, class-level errors, corruption sensitivity, and deployment conversion as part of one analysis.

3 | Materials and Methods

3.1 | Study Design and Task Definition

The experiment was carried out as an experiment that can be reproduced and used as a benchmark for static image classification. For each image, there is one label among the 32 labels provided by the dataset. The study did not involve recognizing sequences of signs, finding word borders, modeling facial grammar or translating entire sentences. For four candidate models, three random seeds were used. Architecture choice was based on the average validation macro-F1 score, while the independent test set was kept for the performance report.

3.2 | Dataset Audit and Duplicate Control

The experiment was carried out as an experiment that can be reproduced and used as a benchmark for static image classification. For each image, there is one label among the 32 labels provided by the dataset. The study did not involve recognizing sequences of signs, finding word borders, modeling facial grammar or translating entire sentences. For four candidate models, three random seeds were used. Architecture choice was based on the average validation macro-F1 score, while the independent test set was kept for the performance report.

3.3 | Dataset Audit and Duplicates Removal

We downloaded the public version of ArSL2018/ArASL from its Kaggle mirror and confirmed its relation to the DOI of the original dataset [1]. The dataset contains 54,049 images belonging to 32 classes. Each image was first hashed using the SHA-256 algorithm in order to find duplicates on the byte level. Then, we computed the 16 x 16 perceptual hash giving us 256 bits of visual signature in order to carry out the second step of auditing the dataset. An image was considered duplicated if its perceptual hash was duplicated in the same class or in different classes.

5,670 duplicates, 243 perceptual hashes repetition within the same classes, and 86 perceptual hashes conflicts between class labels were discovered by the audit. There were images satisfying more than one criterion above; therefore, we cannot sum the numbers. After handling the overlapping cases, 5,956 samples were eliminated and 48,093 images were left. Most importantly, all this was done before creating the train, validation, and test splits, avoiding any possible risk of placing duplicated files in separate splits.

Table 2. Dataset audit and fixed stratified partition profile.

Characteristic	Value
Raw images	54,049
Classes	32
Exact-duplicate flags	5,670
Same-class identical pHash flags	243
Cross-label identical pHash flags	86
Unique records removed after overlapping flags	5,956
Images retained	48,093
Training set	33,665 (70.0%)
Validation set	7,214 (15.0%)
Independent test set	7,214 (15.0%)
Class-size range after audit	1,183-1,917 images

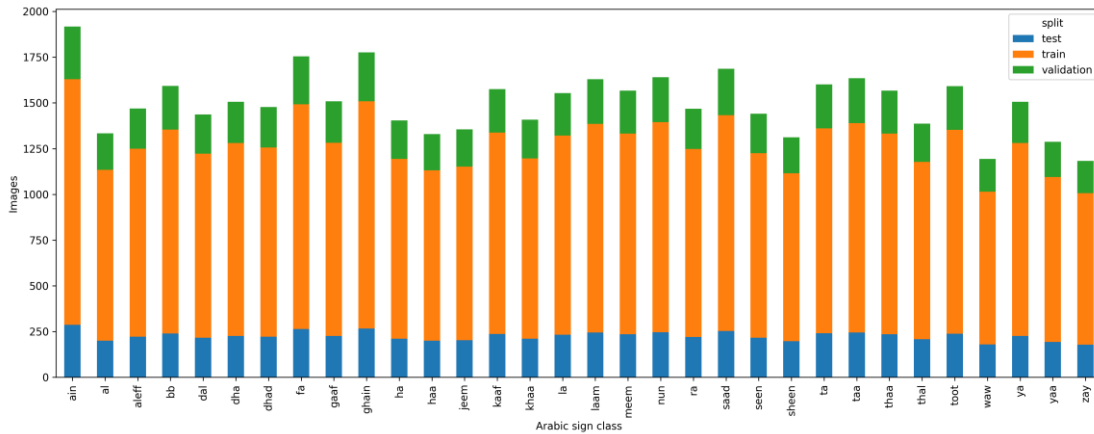


Figure 1. Class distribution across the training, validation, and independent test partitions.

3.4 | Partitioning and Image Preprocessing

The retained images were split in 70/15/15 train/validation/test with a fixed random state. Stratification ensured the same distribution of classes in all the three splits. The images were read as RGB tensors with size 96x96. Class weights were computed from the training split only to ensure that no information about the validation and test splits was used for configuring the training. Data augmentation was performed only on the training images with transformations including: horizontal reflection, rotation by 0.07 of a turn, zoom by 10%, translation by 8% along each spatial axis and contrast variation by 15%.

Horizontal reflection is mentioned explicitly because the mirrored image of the hand might not preserve the meaning of the sign. This data augmentation transformation has already been used in the completed experiment, hence its potential impact on the results is considered as a limitation.

3.5 | Candidate Architectures and Optimization

The benchmark compared four architectures. The custom Lightweight CNN used four depthwise-separable convolutional blocks with 24, 48, 96, and 160 filters. Each block contained batch normalisation, a Swish activation, and max pooling, followed by global average pooling. The remaining models—MobileNetV3-

Small [13], EfficientNetB0 [14], and ResNet50 [15]—were initialised with ImageNet weights after their original classification heads had been removed.

To keep the comparison consistent, every architecture used the same classification head: dropout at 0.30, a 128-unit dense layer with Swish activation, dropout at 0.20, and a 32-class softmax output. The pretrained backbones were frozen during an initial five-epoch stage and optimised with Adam at a learning rate of 0.001. We then unfroze the final 20% of each backbone for three fine-tuning epochs at 0.00001, while leaving batch-normalisation layers frozen. Early stopping monitored validation loss with a patience of two epochs. The learning rate was reduced by a factor of 0.3 after one epoch without improvement, and the batch size was 128.

Table 3. Candidate-model scale and training configuration.

Model	N	Weights	Init.
Light CNN	48.8k	0.49 MB	Random
MobileNet V3	1.02M	9.95 MB	INet
EffNet-B0	4.22M	35.44 MB	INet
ResNet-50	23.85M	181.98 MB	INet

3.6 | Evaluation and Model Selection

For each seed, we evaluated the validation and independent test sets using accuracy, balanced accuracy, macro precision, macro recall, macro-F1, weighted F1, and top-3 accuracy. Results were summarised as the mean and standard deviation across the three seeds. The architecture with the highest mean validation macro-F1 was chosen as the final model family. Within that family, the seed that achieved the strongest validation macro-F1 was used for the confusion matrix, class-level error analysis, corruption tests, and deployment export.

3.7 | Corruption Robustness and Edge Deployment

After model selection, we evaluated the chosen network on the unchanged test set and on five deterministic corruption conditions adapted from common-corruption testing [16]. For the dark and bright conditions, pixel intensities were multiplied by 0.55 and 1.35, respectively. Gaussian noise was added with a standard deviation of 18 intensity units, blur was generated with a 5 x 5 average-pooling kernel, and rotation used a factor of 0.05 of a full turn. These controlled perturbations were not intended to substitute for a real camera study. Their purpose was to reveal obvious weaknesses that would be hidden by clean-image accuracy alone.

The selected network was also converted to Float16 TensorFlow Lite. Quantisation and lightweight inference are widely used to reduce storage and computational demands at deployment time [17]. We recalculated classification performance across the complete test set after conversion. Latency was measured with a single-thread TensorFlow Lite interpreter on 300 test images in the available x86-64 CPU environment. Accordingly, the reported latency should be interpreted as an implementation benchmark rather than as the expected speed of a particular smartphone.

3.8 | Reproducibility and Open Science

The experiments were conducted with Python 3.12.13, TensorFlow 2.20.0, NumPy 2.0.2, pandas 3.0.5, and scikit-learn [18,19]. To support independent inspection, the accompanying repository provides the training script, Kaggle notebook, dependency specification, duplicate audit, fixed split manifest, seed-level results, publication figures, selected weights, Float16 TensorFlow Lite model, and integrity checksums. The original image dataset is not redistributed.

4 | Results

4.1 | Dataset Audit

The duplicate audit removed 11.02% of the original records. After cleaning, individual classes contained between 1,183 and 1,917 images, and stratification preserved those proportions across the training, validation, and test partitions. The independent test set contained 7, 214 images. In other words, the audit changed the effective dataset size substantially before any model was trained.

4.2 | Cross-model Performance

Table 4. Validation-based selection and independent-test performance across three seeds; values in parentheses are standard deviations.

Model	Val. macro-F1	Test accuracy	Test macro-F1
EffNet-B0	0.942 (.002)	0.942 (.002)	0.942 (.002)
ResNet-50	0.916 (.004)	0.917 (.004)	0.916 (.004)
MobileNet V3	0.847 (.001)	0.852 (.002)	0.849 (.002)
Light CNN	0.551 (.069)	0.568 (.039)	0.559 (.046)

The most efficient model in terms of mean validation macro-F1 score (0.9424 ± 0.0016) was EfficientNetB0, which was chosen for detailed analysis of its performance prior to test results presentation. Moreover, the same model won independent-test comparison, having the best mean accuracy (0.9421 ± 0.0021) and mean macro-F1 (0.9420 ± 0.0019). ResNet50 came second but needed about 23.85 million parameters, while EfficientNetB0 needed 4.22 million only. MobileNetV3-Small had a smaller parameter amount but its mean test macro-F1 dropped to 0.8495. The custom Lightweight CNN was the smallest one but less accurate and more sensitive to the random seed choice.

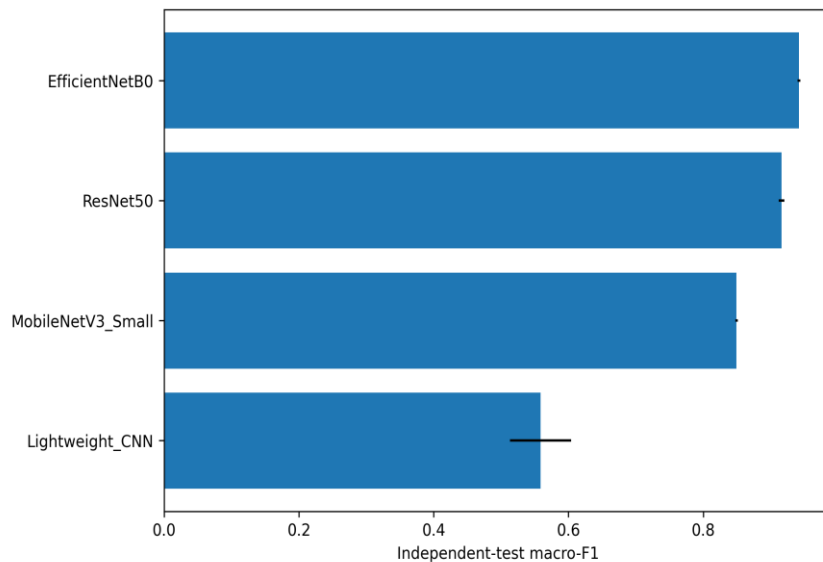


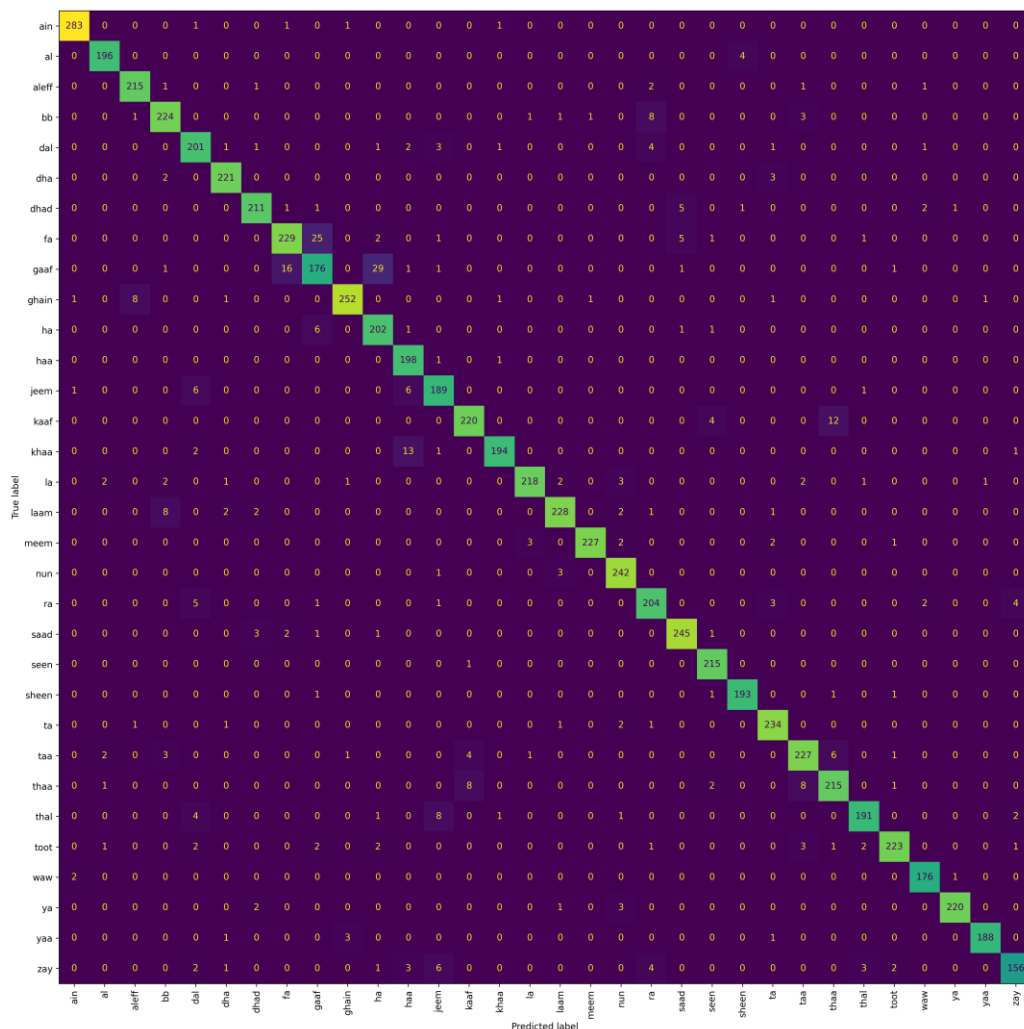
Figure 2. Mean independent-test macro-F1 per model; error bars represent standard deviation per three seeds.

Random forest and k-nearest neighbours both reached a slightly higher raw accuracy of 92.86%. However, each model misclassified two of the eight negative cases, which reduced specificity to 75% and balanced accuracy to 85.42%. RBF-SVM achieved the highest ROC-AUC (0.977) and the lowest Brier score (0.051), although its sensitivity was lower than that of logistic regression. This pattern shows why the model-selection rule should be defined on the development data rather than choosing whichever model happens to have the most attractive test-set metric.

4.3 | Selected-run Errors and Class-level Performance

Out of the three EfficientNetB0 experiments, that of seed 42 scored the highest validation macro-F1 and thus is chosen for the analysis of errors. In terms of results on the independent test set without corruption, it scored 94.44% accuracy, 94.46% balanced accuracy, 94.46% macro precision, 94.46% macro recall, 94.42% macro-F1, and 98.88% top-3 accuracy.

However, there was no uniformity across all classes. The F1-score varied from 0.8018 for gaaf to 0.9861 for ain. The most frequent directed error type is gaaf recognized as ha, observed 29 times. After that is fa identified as gaaf, with 25 errors, and finally gaaf identified as fa, with 16 errors. This shows that visually similar hand positions continue to pose difficulty for the model. Figure 3. Confusion matrix for the validation-selected EfficientNetB0 seed-42 model.



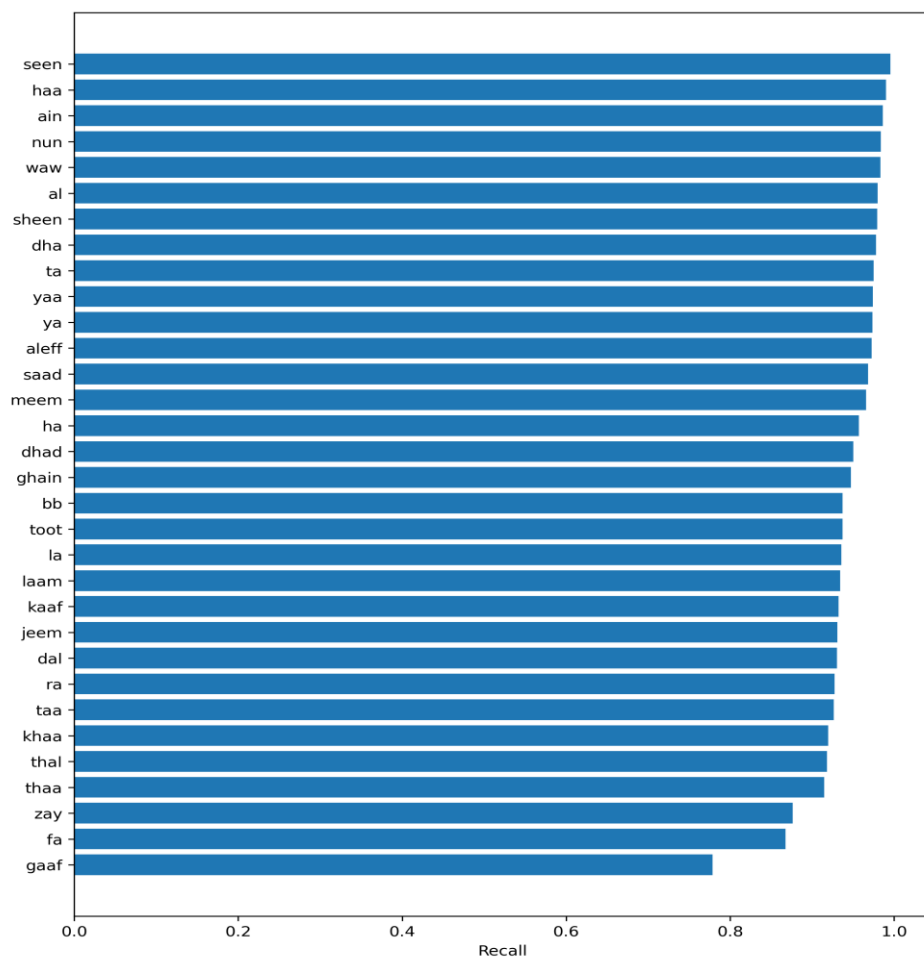


Figure 3. Per-class recall for the selected EfficientNetB0 model.

4.4 | Robustness under Common Image Corruptions

Table 5. Selected-model performance under deterministic image corruptions.

Condition	Acc.	Macro-F1	F1 drop (pp)
Clean	0.944	0.944	0.00
Dark	0.927	0.926	1.78
Bright	0.894	0.893	5.12
Noise	0.747	0.745	19.96
Blur	0.904	0.904	4.05
Rotation	0.938	0.937	0.70

The least impact on macro-F1 was made by small rotations; macro-F1 was reduced by only 0.70%. The darkest images were reduced by 1.78%, but blur and brightening images were associated with the greatest reduction in macro-F1 scores, which were 4.05% and 5.12%, respectively. Clearly, Gaussian noise was the condition that had the most negative impact on performance: macro-F1 was reduced by 19.96% to 0.7446. Still, top-3 accuracy was greater than 0.93 even under this condition.

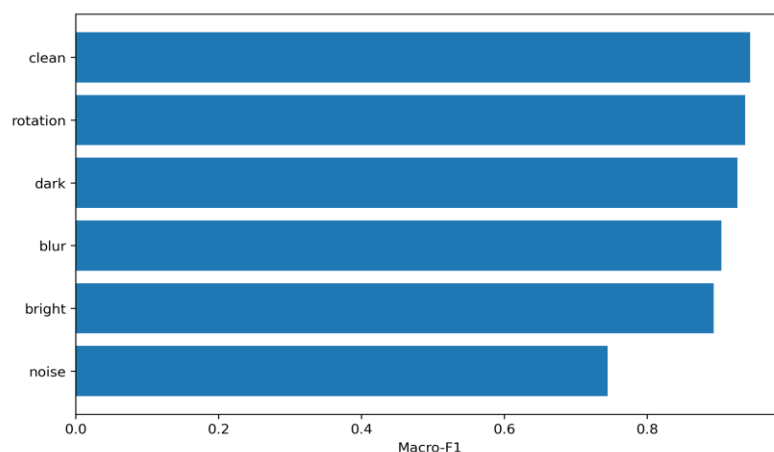


Figure 4. Macro-F1 of the selected model on clean and corrupted test images.

4.5 | TensorFlow Lite Deployment

EfficientNetB0 model weight file selected took up space of 35.44 MB. The conversion to TensorFlow Lite using Float16 format made the model lighter by reducing the size to 8.03 MB, a reduction of 77.34%. After the conversion, 93.14% of accuracy and 93.14% of macro-F1 were maintained, which corresponds to a reduction of 1.30 percentage points in accuracy compared to the accuracy of the clean Keras model. The top-3 accuracy was 98.75%. The median latency for single-threaded CPU was 4.12 milliseconds, equivalent to 242.

5 | Discussion

5.1 | Principal Findings

Several key points can be identified. First, duplicate auditing had a significant impact on the dataset: 5,956 images, which constituted one record in nine, were excluded prior to the partitioning process. Second, EfficientNetB0 performed the best compromise between accuracy and model complexity. Lastly, the study of model robustness showed why the accuracy on the clean validation set could not be considered a fully representative metric of the model's preparedness for deployment. The model was rather tolerant to rotation and decreased illumination, but suffered from additive noise.

Lastly, the fact that several random seeds were used provided an additional insight into the matter. While EfficientNetB0 demonstrated the best average score together with small standard deviation, which means that the model is relatively tolerant to the choice of initialization parameters, the variance of Lightweight CNN was rather high. Therefore, describing only the best result of this model would have been incorrect.

5.2 | Relationship to Earlier Arabic Sign Recognition Studies

With the accuracy of 94.21%, which the EfficientNetB0 demonstrates, the present model falls short of a number of recent advances in recognition of Arabic signs [7,10,11]. This does not mean, however, that the previous models were worse or that the present model is better from the perspective of methodology. These studies kept a different number of images, used different resolution and augmentation strategies, balanced differently and selected differently, and even sometimes considered a different problem. Thus, CLIP-ArASL worked with a balanced dataset of 400 images per class along with bilingual text as the semantic anchor [11]. Ismaiel et al. considered recognition enhanced by the Canny edges information [10]. ArSL-EdgeBench went another way by focusing on the disclosure of duplicates, three-seed recognition, selection for validation only, corruption and edge-oriented exporting.

This distinction becomes especially relevant as the accuracy results of Alyami et al. show 99.74% in signer-dependent conditions and only 68.2% when the test phase used unseen signers [8]. As the publicly available ArSL2018 dataset lacks information regarding signer identity, the achieved 94.44% accuracy should be considered image-level generalization without duplicates. Thus, this result cannot be taken as evidence of similar accuracy in individuals not seen during training.

5.3 | Practical Implications

From the results with Float16, EfficientNetB0 can be compressed down to quite small storage size without sacrificing most of its test accuracy. However, there would still be much needed in order to produce a usable mobile system. These include hand detection, normalization of the camera, confidence threshold, temporal smoothing, and interface which gives an opportunity to manually fix ambiguous cases. Outputting top-3 would be beneficial in this interface, because correct class was present in the top 3 predictions for almost 99% of clean images and more than 93% of noisy ones.

Recognition and communication tasks should be treated separately. Assignment of alphabet labels to each frame does not mean that Arabic Sign Language is translated into text. A spelling interface has to be able to process temporal segmentation, repetition of letters, and transitions between signs. Translation requires further modelling of gestures, lexicographic context, facial grammar, and linguistics.

5.4 | Limitations

The study should be interpreted in light of several limitations. First, signer identifiers are absent from the public dataset, so a signer-independent split was not possible. Second, the images are static and largely centred, unlike the unconstrained sequences produced by a mobile camera. Third, the duplicate screen used identical perceptual hashes as a conservative rule; visually similar images with different hashes may still remain. Fourth, horizontal reflection was used during augmentation even though mirroring may alter the meaning of some hand configurations. A dedicated ablation is needed before that transformation can be considered linguistically safe.

The robustness analysis also used synthetic corruptions at a single severity level. It cannot replace prospective evaluation with new signers, varied skin tones, unfamiliar backgrounds, different devices and camera distances, natural lighting, and realistic occlusions. In addition, latency was measured on an x86-64 Kaggle/Colab-class CPU rather than on a named mobile processor. Finally, only three random seeds were evaluated. The reported standard deviations are therefore descriptive, and the experiment was not designed to support a strong statistical significance claim.

6 | Conclusion

ArSL-EdgeBench offers a transparent, duplicate-controlled comparison of four neural architectures for static Arabic sign alphabet recognition. Cleaning the original ArSL2018 release removed 5,956 records before partitioning, after which EfficientNetB0 produced the strongest validation and test results across three random seeds. The selected run achieved 94.44% accuracy on the clean independent test set. Float16 TensorFlow Lite conversion reduced the model to 8.03 MB while preserving 93.14% accuracy. Although the network remained stable under modest rotation and reduced illumination, Gaussian noise caused a substantial loss in performance. EfficientNetB0 is therefore a useful baseline for static alphabet prototypes, but the benchmark should not be presented as continuous sign-language translation or as evidence of unseen-signer generalisation. The most valuable next step is a signer-disjoint camera study that uses linguistically safe augmentation and temporal smoothing.

Acknowledgments

The authors thank the creators of ArSL2018 for making the dataset publicly available and acknowledge the developers and maintainers of the open-source Python and TensorFlow software used in this work.

Author Contribution

All authors contributed equally to this research

Funding

This research received no external funding.

Institutional Review Board Statement

No new human participants were recruited or contacted for this study. The analysis was limited to a previously published, publicly accessible image dataset, and institutional review-board approval was therefore not sought. This wording should be checked against the requirements of the authors' final institutions before submission.

Informed Consent Statement

Not applicable to this secondary computational analysis. Information about consent for the original data collection is provided by the dataset authors [1].

Data Availability

ArSL2018 is publicly available through Mendeley Data under DOI 10.17632/y7pckrw6z2.1 and is described in [1]. The original images are not included with this manuscript. The repository provides the duplicate audit and the fixed split manifest needed to reproduce the reported partitions.

Code Availability

Source code, the Kaggle notebook, dependency specifications, result tables, figures, sanitized manifests, selected model weights, and the Float16 TensorFlow Lite model are available in the [ArSL-EdgeBench GitHub repository](#) [20].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

- [1] Latif, G., Mohammad, N., Alghazo, J., AlKhalaf, R., & AlKhalaf, R. (2019). ArASL: Arabic alphabets sign language dataset. *Data in Brief*, 23, 103777. <https://doi.org/10.1016/j.dib.2019.103777>
- [2] Alayed, A. (2024). Machine learning and deep learning approaches for Arabic Sign Language recognition: A decade systematic literature review. *Sensors*, 24(23), 7798. <https://doi.org/10.3390/s24237798>
- [3] Al-Shamayleh, A. S., Ahmad, R., Jomhari, N., & Abushariah, M. A. M. (2020). Automatic Arabic Sign Language recognition: A review, taxonomy, open challenges, research roadmap and future directions. *Malaysian Journal of Computer Science*, 33(4), 306-343. <https://doi.org/10.22452/mjcs.vol33no4.5>
- [4] Sidig, A. A. I., Luqman, H., Mahmoud, S., & Mohandes, M. (2021). KArSL: Arabic Sign Language database. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 20(1), Article 14. <https://doi.org/10.1145/3423420>
- [5] Abbas, S., Alahmadi, D., & Al-Barhamtoshy, H. (2024). Establishing a multimodal dataset for Arabic Sign Language production. *Journal of King Saud University - Computer and Information Sciences*, 36(8), 102165. <https://doi.org/10.1016/j.jksuci.2024.102165>
- [6] Tharwat, G., Ahmed, A. M., & Bouallegue, B. (2021). Arabic Sign Language recognition system for alphabets using machine learning techniques. *Journal of Electrical and Computer Engineering*, 2021, 2995851. <https://doi.org/10.1155/2021/2995851>

- [7] Alharthi, N. M., & Alzahrani, S. M. (2023). Vision transformers and transfer learning approaches for Arabic Sign Language recognition. *Applied Sciences*, 13(21), 11625. <https://doi.org/10.3390/app132111625>
- [8] Alyami, S., Luqman, H., & Hammoudeh, M. (2024). Isolated Arabic Sign Language recognition using a transformer-based model and landmark keypoints. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 23(1), Article 3. <https://doi.org/10.1145/3584984>
- [9] Noor, T. H., Noor, A., Alharbi, A. F., Faisal, A., Alrashidi, R., Alsaedi, A. S., Alharbi, G., Alsanoosy, T., & Alsaedi, A. (2024). Real-time Arabic Sign Language recognition using a hybrid deep learning model. *Sensors*, 24(11), 3683. <https://doi.org/10.3390/s24113683>
- [10] Ismaiel, W., Chaieb, C., Aribi, Y., Dawood, O., & Merghani, W. (2026). Leveraging edge detection techniques to enhance Arabic Sign Language static-gesture recognition using deep learning. *Journal of Engineering Research*, 14(1), 896-915. <https://doi.org/10.1016/j.jer.2025.09.011>
- [11] Alasmari, N. (2026). CLIP-ArASL: A lightweight multimodal model for Arabic Sign Language recognition. *Applied Sciences*, 16(5), 2573. <https://doi.org/10.3390/app16052573>
- [12] Kapoor, S., & Narayanan, A. (2023). Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9), 100804. <https://doi.org/10.1016/j.patter.2023.100804>
- [13] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for MobileNetV3. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1314-1324. <https://doi.org/10.1109/ICCV.2019.00140>
- [14] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning*, 97, 6105-6114.
- [15] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770-778. <https://doi.org/10.1109/CVPR.2016.90>
- [16] Hendrycks, D., & Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *International Conference on Learning Representations*. <https://arxiv.org/abs/1903.12261>
- [17] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2704-2713. <https://doi.org/10.1109/CVPR.2018.00286>
- [18] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- [19] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 265-283.
- [20] ArSL-EdgeBench contributors. (2026). ArSL-EdgeBench: Code, results, and edge-model artifacts. GitHub repository. [Repository link](#)

Disclaimer/Publisher's Note: The perspectives, opinions, and data shared in all publications are the sole responsibility of the individual authors and contributors, and do not necessarily reflect the views of Sciences Force or the editorial team. Sciences Force and the editorial team disclaim any liability for potential harm to individuals or property resulting from the ideas, methods, instructions, or products referenced in the content.