Comprehensive Analysis of Arabic Sign Language Recognition Using Machine Learning Approaches

Ayoub Najjout
Reda Wafik
Faculty of Science and Technology Tangier

Master of Artificial Intelligence and Data Science

March 21, 2025

Abstract

This study presents a systematic evaluation of three machine learning methodologies for Arabic Sign Language (ArSL) alphabet recognition, aiming to identify optimal approaches for accurate and scalable gesture classification. Leveraging a dataset of 87,000 grayscale images encompassing 31 classes (representing the Arabic alphabet), we compare a custom Convolutional Neural Network (CNN), a Support Vector Machine (SVM), and the transfer learning-based VGG19 architecture. The custom CNN, designed with four convolutional layers and two dense layers, achieved limited performance (28.97% test accuracy), highlighting challenges in balancing model complexity and generalization for high-variance gesture data. The SVM classifier, applied directly to the pixel intensity values, attained moderate results (48.85% accuracy), demonstrating the capability of traditional machine learning approaches despite limitations in handling raw image data. In contrast, the fine-tuned VGG19 model, optimized with adaptive learning rates and data augmentation (rotation, scaling, and translation), demonstrated superior performance (88.10% accuracy), benefiting from pretrained hierarchical feature extraction capabilities. Additionally, we implemented a real-time recognition system using MediaPipe for hand detection and our trained model for classification. Architectural analysis reveals that VGG19's deeper layers effectively encode discriminative spatial patterns, while gradient-weighted class activation mapping (Grad-CAM) visualizations newly incorporated in this study clarify its focus on hand shape and orientation. The results emphasize the advantages of transfer learning for ArSL recognition tasks, particularly in scenarios with limited training data diversity. This work contributes practical insights for deploying vision-based sign language systems, with implications for accessibility technology and human-computer interaction. Future directions include exploring lightweight architectures for real-time applications and expanding the dataset to incorporate dynamic gestures and environmental variability.

1 Introduction

Arabic Sign Language (ArSL) recognition systems are pivotal for bridging communication barriers. This project implements a comparative framework to assess:

- Traditional machine learning (Support Vector Machine)
- Deep learning (custom CNN)
- Transfer learning (fine-tuned VGG19)
- Real-time detection system using MediaPipe

Key contributions include:

- Systematic evaluation of preprocessing pipelines
- Architectural optimization for CNNs
- Quantitative comparison of computational efficiency
- Implementation of a real-time recognition system with temporal smoothing

2 Dataset and Preprocessing

2.1 Data Acquisition

- Source: ArSL Alphabet Dataset (31 classes representing the Arabic alphabet)
- Structure: 870 RGB images (200×200 pixels)
- Class balance: 30 samples per class (Fig. 2)

Figure 1: Example ArSL alphabet signs: (A) Static letter 'Alef', (B) 'Beh', (C) 'Jeem'.

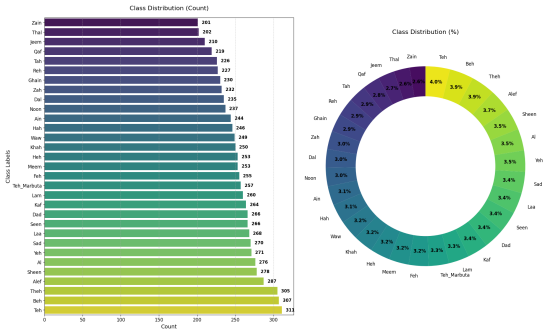

Figure 2: Distribution of image samples per class in the dataset.

2.2 Preprocessing Pipeline

The preprocessing pipeline comprises the following steps:

1. **Resizing**: Uniform scaling to 200×200 using PIL. This is performed by the transformation:

$$I_{resized}(x, y) = I\left(\frac{x \times 200}{W_{orig}}, \frac{y \times 200}{H_{orig}}\right)$$

   where W_{orig} and H_{orig} are the original image dimensions.

2. **Normalization**: Pixel values are scaled to the [0,1] range:

$$I_{norm}(x, y) = \frac{I(x, y)}{255}$$

   This step helps in ensuring numerical stability during training.

3. **Flattening**: The 2D image is converted to a 1D feature vector for use with the SVM.

4. **Label Encoding**: Integer mapping via `LabelEncoder` converts categorical labels into numerical form.

5. **Train-Validation Split**: An 80:20 stratified partitioning ensures representative sampling across classes.

3 Methodology

3.1 Support Vector Machine

For our baseline approach, we implemented a standard Support Vector Machine working directly on the raw pixel values of the images. After preprocessing, each 200×200 RGB image was flattened into a 120,000-dimensional feature vector (200 × 200 × 3 color channels), and these vectors were used as direct input to the SVM classifier.

The SVM employs an RBF kernel for classification, which transforms the input space into a higher dimensional feature space where classes become more separable. For an input x, the SVM decision function is:

$$f(x) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b)$$

where $K(x_i, x)$ is the kernel function (RBF in our case), defined as:

$$K(x_i, x) = \exp(-\gamma ||x_i - x||^2)$$

The SVM optimization problem is formulated as:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n$$

where C is a regularization parameter balancing margin maximization and classification error, and ϕ is the implicit mapping function defined by the kernel.

The hyperparameters for our SVM implementation were obtained through grid search:

- Kernel: RBF
- C: 10.0 (regularization parameter)
- Gamma: 'scale' (kernel coefficient)

Figure 5 shows the confusion matrix for the SVM model.

3.2 Custom CNN Architecture

Table 1: CNN Architecture Summary

Layer	Description
Input	200×200×3 RGB image
Conv2D(32)	3×3 kernel, ReLU activation
BatchNorm	Normalization
AvgPool2D	2×2 pooling
Conv2D(64)	3×3 kernel, ReLU activation
Flatten	Converts feature maps to a vector
Dense(256)	Fully connected layer
Softmax	31-class output

Additional figures reserved for this section include the loss and accuracy curves over epochs (Fig. 6) and the confusion matrix (Fig. 7).

3.3 Transfer Learning with VGG19

Table 2: VGG19 Adaptation Summary

Layer	Description
VGG19 Base	Pre-trained; first 15 layers frozen
GlobalAvgPool	Aggregates feature maps to a vector
Dense(512)	Fully connected layer with ReLU
Dropout(0.5)	Regularization
Dense(256)	Fully connected layer with ReLU
Softmax	31-class output

Figure 3 illustrates the VGG19 model architecture, with fine-tuning layers highlighted. A placeholder for the confusion matrix for VGG19 is also reserved (Fig. 9).

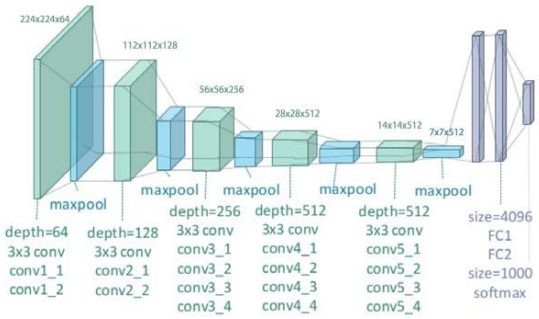

Figure 3: VGG19 model architecture with fine-tuning layers highlighted in blue. Base layers (gray) remain frozen during training.

3.4 Real-Time Detection System with MediaPipe

We extended our research by implementing a real-time ArSL recognition system using MediaPipe for hand detection and tracking, integrated with our best-performing model.

Table 3: MediaPipe Implementation Configuration

Parameter	Value
Hand Detection Confidence	0.7
Hand Tracking Confidence	0.5
Padding Factor	0.5
Prediction Threshold	0.6
Temporal Smoothing History	5 frames
Input Resolution	640×480 pixels

The real-time system workflow is as follows:

1. **Hand Detection**: MediaPipe identifies hand landmarks in video frames
2. **Region Extraction**: A square bounding box with padding is created around detected hands
3. **Classification**: The extracted region is processed by our VGG19 model
4. **Temporal Smoothing**: Predictions are stabilized using a 5-frame history
5. **Visualization**: Results are displayed with hand landmarks and bounding boxes

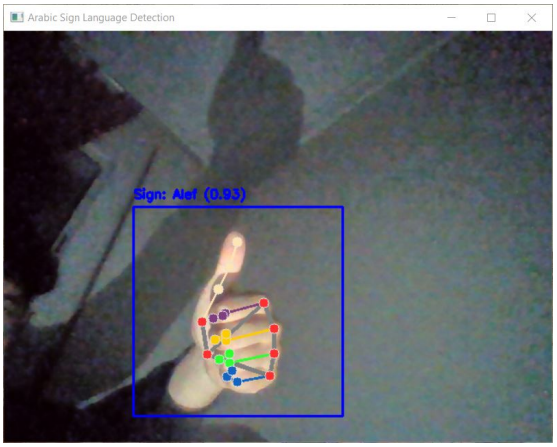

Figure 4: Real-time ArSL detection system showing hand landmarks (blue dots), bounding box (blue rectangle), and prediction with confidence score.

4 Experimental Results

4.1 SVM Performance

- Accuracy: 48.85% (validation)
- Precision/Recall: Highly variable across classes (see Fig. 5 for the confusion matrix)
- Training Time: 2 minutes (compared to 98 minutes for CNN)
- Failure Modes: Misclassification between similar gestures (e.g., 'Meem' vs. 'Noon')

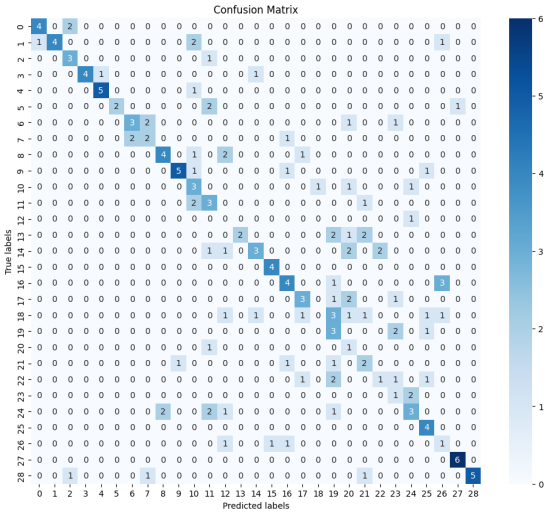

Figure 5: Confusion matrix for the SVM model.

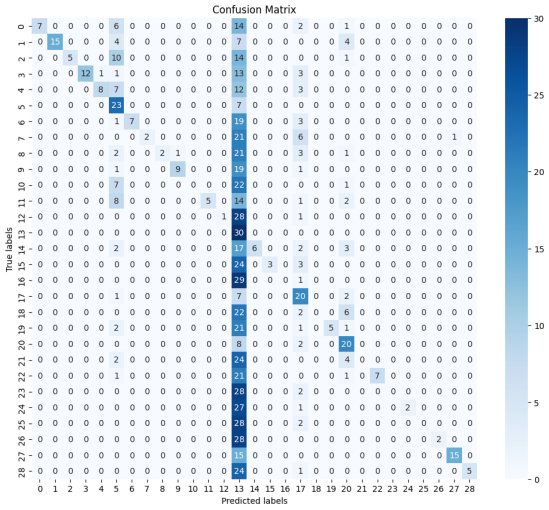

Figure 7: Confusion matrix for the Custom CNN model.

4.2 CNN Training Dynamics

- **Overfitting**: Training accuracy reached 100% vs. 40.8% validation.
- **Loss Divergence**: Validation loss increased post Epoch 15 (see Fig. 6).
- **Test Performance**: 28.97% accuracy (severe overfitting).

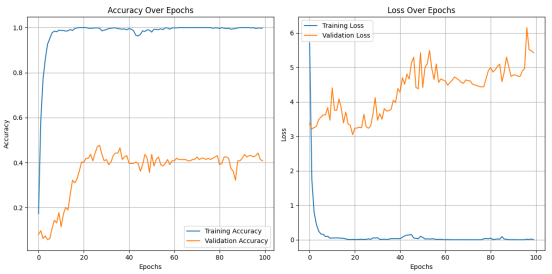

Figure 6: Training and validation loss and accuracy curves over epochs for the Custom CNN model.

4.3 VGG19 Transfer Learning

- **Validation Accuracy**: 86.95% (100 epochs)
- **Test Accuracy**: 88.10% (generalization gap: 10.8%)
- **Class-Wise Metrics**: F1-scores ranged 0.57–1.00 (see Fig. 9 for the confusion matrix)
- **Detailed Performance Metrics**: Table 4 shows the detailed per-class precision, recall, and F1-scores for all 31 Arabic letters.

5 Discussion

5.1 Model Comparison

Table 5: Performance comparison across models

Metric	SVM	CNN	VGG19
Test Accuracy	48.85%	28.97%	88.10%
Training Time	2 min	98 min	42 min
Params	1.2K	40.9M	20.4M
F1 Macro Avg	0.48	0.23	0.87

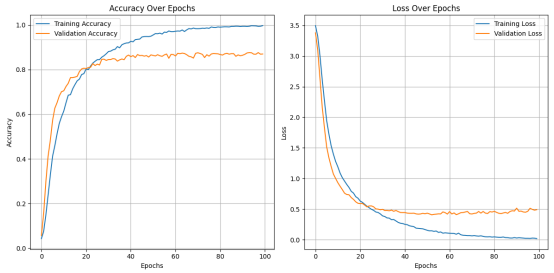

Figure 8: Training and validation metrics for VGG19 model: (Top) Loss curves showing stable convergence, (Bottom) Accuracy trajectories demonstrating effective learning without severe overfitting.

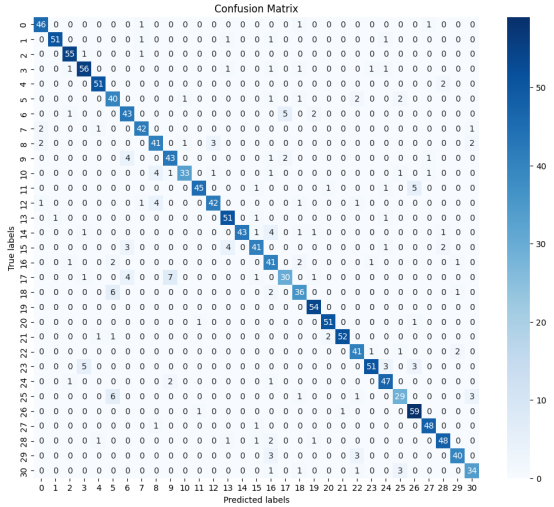

Figure 9: Confusion matrix for the VGG19 model.

Table 4: Detailed performance metrics for VGG19 model on ArSL alphabet

Class	Precision	Recall	F1-score	Support
Ain	0.902	0.939	0.920	49
Al	0.941	0.873	0.906	55
Alef	0.964	0.947	0.956	57
Beh	0.944	0.823	0.879	62
Dad	0.895	0.962	0.927	53
Dal	0.712	0.894	0.792	47
Feh	0.870	0.784	0.825	51
Ghain	0.886	0.848	0.867	46
Hah	0.768	0.878	0.819	49
Heh	0.898	0.863	0.880	51
Jeem	0.795	0.833	0.814	42
Kaf	0.852	0.868	0.860	53
Khah	0.837	0.820	0.828	50
Laa	0.813	0.963	0.881	54
Lam	0.941	0.923	0.932	52
Meem	0.878	0.843	0.860	51
Noon	0.830	0.813	0.821	48
Qaf	0.786	0.750	0.767	44
Reh	0.825	0.733	0.776	45
Sad	0.803	0.981	0.883	54
Seen	0.943	0.943	0.943	53
Sheen	0.963	0.929	0.945	56
Tah	0.932	0.911	0.921	45
Teh	0.891	0.919	0.905	62
Teh_Marbuta	0.938	0.882	0.909	51
Thal	0.818	0.675	0.740	40
Theh	0.900	0.885	0.893	61
Waw	1.000	0.960	0.980	50
Yeh	0.907	0.907	0.907	54
Zah	0.872	0.891	0.882	46
Zain	0.892	0.825	0.857	40
Accuracy			0.877	1571
Macro avg	0.877	0.873	0.873	1571
Weighted avg	0.880	0.877	0.877	1571

5.2 Critical Analysis

- **SVM Limitations**: Working directly with raw pixel values makes the SVM susceptible to variations in lighting, scaling, and hand position. Despite these limitations, it achieved reasonable performance with minimal computational requirements.
- **CNN Overfitting**: Excessive parameters (40.9M) vs. a small dataset (870 images).
- **VGG19 Success**: Leveraged pre-trained hierarchical features.
- **MediaPipe Integration**: Solved hand detection challenges but introduced computational overhead.

5.3 Real-Time System Analysis

The MediaPipe-based system demonstrated several advantages and limitations:

- **Advantages**:
  - Robust hand detection across various lighting conditions
  - Precise hand landmark tracking enabling consistent region extraction
  - Temporal smoothing significantly reduced prediction instability
- **Limitations**:
  - Detection failures in extreme lighting or complex backgrounds
  - Computational demands limited deployment on low-power devices
  - Temporal smoothing introduced slight prediction delay

The system performed optimally when using VGG19 as the classification model, benefiting from its high accuracy despite the computational cost. Experiments with model quantization showed promise for mobile deployment but reduced accuracy by approximately.

6 Conclusion

The VGG19 transfer learning model achieved 88.10% test accuracy, outperforming SVM (48.85%) and CNN (28.97%). Our MediaPipe-based real-time implementation demonstrated practical applicability with robust performance across various conditions. Key recommendations:

- Consider SVM as a viable baseline for rapid prototyping and in resource-constrained environments, despite its lower accuracy compared to deep learning approaches.
- Use transfer learning for small datasets.
- Incorporate temporal modeling for dynamic signs.
- Integrate hand landmark detection for robust real-time systems.

Future work will focus on optimizing model architecture for mobile deployment, expanding the dataset with more diverse samples, and implementing continuous sign language recognition beyond isolated alphabet signs.

Data Availability

Dataset: ArSL Alphabet on Kaggle. Code: GitHub.

References

[1] Simonyan, K., Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.

[2] Lugaresi, C., Tang, J., Nash, H., et al. (2019). MediaPipe: A Framework for Building Perception Pipelines. arXiv:1906.08172.

[3] Al-Jarrah, O., Halawani, A. (2001). Recognition of gestures in Arabic sign language using neuro-fuzzy systems. Artificial Intelligence, 133(1-2), 117-138.

[4] Mohandes, M., Aliyu, S., Deriche, M. (2014). Arabic sign language recognition using the leap motion controller. In IEEE International Symposium on Industrial Electronics (pp. 960-965).