A Comparative Study on Performance of Pretrained ImageNET Models for Character Level Static Hand Gesture Recognition

Adib Sakhawat, Md Hasibur Rahman, Minhajul Abedin, Nabila Islam, Nazifa Tasneem
BSc in Software Engineering
Department of CSE, Islamic University of Technology
Gazipur, Bangladesh
adibsakhawat@iut-dhaka.edu, hasiburrahman21@iut-dhaka.edu, minhajul@iut-dhaka.edu,
nabilaislam21@iut-dhaka.edu, nazifatasneem@iut-dhaka.edu

***Abstract*—The challenges faced by hearing and speech-impaired individuals in utilizing modern voice and speech-based systems necessitate the development of Human-Computer Interaction solutions based on sign language recognition systems. This study conducts a comparative analysis of 15 *Convolutional Neural Network* (CNN) models under controlled hardware, software, and environmental conditions to evaluate their performance on the Sign Language Gesture Images Dataset [13]. The models' performance is assessed using various metrics, including *Top-1 Accuracy*, with ConvNeXtXLarge achieving the highest accuracy of 99.63%. This work highlights the potential of CNN-based approaches in advancing sign language recognition for improved accessibility.**

***Index Terms*—Static Hand Gesture Recognition, Deep Learning, Pretrained Models, Transfer Learning, Sign Language Recognition, Human-Computer Interaction, Computer Vision, Image Classification, Model Interpretability, Evaluation Metrics**

I. Introduction

In the context of human-computer interaction (HCI) and gesture-based interfaces, static hand gesture recognition plays an important role. It enables non-verbal, touch-free communication. These systems, based on machine learning models such as convolutional neural networks (CNNs), are becoming increasingly relevant for intuitive and efficient communication methods [1], [3], [9].

Pre-trained models, such as those trained on ImageNet, provide robust feature extraction capabilities. It makes them highly effective for domain-specific tasks like gesture recognition. Their ability to transfer knowledge has demonstrated improved accuracy even with smaller gesture datasets [7], [9]. CNNs have revolutionized gesture recognition by outperforming traditional algorithms, such as Support Vector Machines (SVMs), due to their superior ability to model complex spatial relationships in images [2], [3].

This study presents a comparative analysis of pre-trained ImageNet models for static hand gesture recognition. It focuses on their accuracy, efficiency, and real-world applicability. The findings aim to advance gesture-based HCI systems. These address key challenges in developing reliable communication interfaces.

II. Literature Review

Static hand gesture recognition has emerged as a critical component in Human-Computer Interaction (HCI). It facilitates non-verbal communication through machine learning and deep learning techniques. Over the years, substantial research has been conducted to address the challenges of accurate classification, computational efficiency, and real-world applicability of gesture recognition systems.

A. Static Hand Gesture Recognition Systems

Static hand gesture recognition focuses on identifying fixed hand postures captured in single images or frames. It has significant applications in sign language recognition, virtual environments, and assistive technologies. Early systems relied on traditional machine learning methods like Support Vector Machines (SVMs) and Hidden Markov Models (HMMs). These required manual feature extraction and often suffered from low generalization to real-world conditions [2], [3]. The shift towards vision-based approaches using deep learning, particularly CNNs, marked a breakthrough in gesture recognition. CNNs automate feature extraction through hierarchical layers. It does so by enabling robust classification performance. For example, a comparative study showed that CNNs outperformed SVM-based classifiers for Tanzanian Sign Language Recognition, achieving superior accuracy due to their ability to model spatial features effectively [2]. However, challenges such as intraclass variability, background noise, and illumination changes remain. Studies emphasize the need for optimized architectures and pre-trained models to improve performance, especially in constrained datasets [3], [9].

B. Role of Convolutional Neural Networks (CNNs)

CNNs have been widely adopted for static hand gesture recognition due to their ability to extract spatial and structural features. A study analyzing static ASL alphabet recognition compared multiple CNN architectures optimized with different activation functions and optimizers. Results revealed that PReLU activation with the SGDM optimizer achieved an accuracy of 99.3%, outperforming RMSProp-based models, which only reached 83.3% accuracy [1]. This highlights the impact of activation functions and optimization strategies in achieving high performance. Another comparative study on neural network architectures emphasized the importance of model size and connectivity [4]. Fully Connected Cascade (FCC) networks and Bridged Multilayer Perceptrons (BMLP) demonstrated significant improvements over standard Multilayer Perceptrons (MLPs). FCC achieved higher accuracy with fewer neurons, indicating the need for efficient architectures to balance performance and computational cost [4]. In addition, hybrid models combining CNNs with traditional classifiers, such as HMMs, were found to improve recognition accuracy for dynamic gestures, further bridging the gap between static and continuous gesture recognition tasks [3].

C. Pretrained ImageNet Models for Gesture Recognition

Pretrained models have revolutionized deep learning by enabling transfer learning—leveraging knowledge from large datasets like ImageNet to solve domain-specific tasks. ImageNet, a benchmark dataset with over 3.2 million labeled images across 5247 categories, has played an important role in developing state-of-the-art models for object recognition [7].

For static hand gesture recognition, pretrained ImageNet models provide significant advantages:

1) **Feature Generalization:** ImageNet-trained CNNs, such as ResNet and VGG, offer robust feature extractors that generalize well to new tasks [7].
2) **Reduced Training Time:** By fine-tuning pretrained models, researchers can achieve high accuracy with limited computational resources [9].
3) Improved Accuracy: Studies highlight the success of pretrained networks in overcoming challenges like noise, occlusion, and varying hand poses [8], [9].

The relevance of pretrained ImageNet models is further underscored by their performance on sign language datasets, where they achieved superior accuracy compared to networks trained from scratch. For example, fine-tuning ResNet on the Sign Language Gesture Dataset showed improved recognition performance, validating the importance of pretrained models in small-data scenarios [12], [13].

D. Activation Functions and ReLU-Based Networks

The choice of activation functions significantly impacts the performance and efficiency of neural networks. ReLU (Rectified Linear Unit) has become the preferred activation function for CNNs due to its computational simplicity and ability to mitigate the vanishing gradient problem. A theoretical study demonstrated that ReLU-based deep neural networks (DNNs) are well-suited for representing continuous piecewise linear (CPWL) functions, which are integral to vision tasks like hand gesture recognition [5]. ReLU networks require fewer layers compared to alternative activations while maintaining high representation power. For tasks involving spatial relationships, such as static hand gesture classification, this makes ReLU-based CNNs highly efficient and accurate [5]. Additionally, studies show that combining ReLU networks with graph-based models or hybrid architectures can further improve performance by capturing complex relationships between features [6].

E. Challenges in Static Hand Gesture Recognition

Despite the success of CNNs and pretrained models, static hand gesture recognition still faces several challenges:

1) **Dataset Limitations:** Existing gesture datasets are often constrained by small sample sizes, regional biases, and limited variations in poses and lighting conditions [12], [13].
2) **Model Optimization:** Selecting the appropriate neural network size and architecture is critical for balancing generalization and computational efficiency [4].
3) **Real-World Variability:** Robustness to noise, occlusion, and cluttered backgrounds remains a challenge for real-world deployment of gesture recognition systems [3].

Researchers have proposed solutions such as graph-based learning and hybrid models to address these challenges. For example, integrating CNNs with Graph Neural Networks (GNNs) allows for spatial-temporal feature extraction, improving accuracy for both static and dynamic gestures [6].

III. Dataset Preparation and Preprocessing

This study employs the *Sign Language Gesture Images Dataset* sourced from Kaggle [13]. The dataset comprises 55,500 images categorized into 37 distinct gesture classes, including alphanumeric labels such as R, A, and 7. Each class is organized into separate subdirectories, and all images are stored in `.jpg` format with uniform dimensions of 50×50 pixels.

A. Dataset Structure and Integrity

Each subdirectory corresponds to a specific gesture class, ensuring organized storage and easy accessibility. An integrity check revealed no corrupted files, affirming the high quality of the dataset [14]. Additionally, the dataset contains 12,165 duplicate images, which were identified and flagged during preprocessing to maintain data quality [14].

B. Data Preprocessing

The images were loaded with `cv2.imread` and verified to have the shape $(50, 50, 3)$ [14]. The intensity values of the pixels were normalized to the range [0, 1] by dividing by 255.0, standardizing the input for subsequent model training. Two label mappings were established to facilitate efficient numerical indexing: a direct class-to-index mapping and its reverse.

C. Data Augmentation

To enhance data set diversity and improve model generalization, `ImageDataGenerator` from TensorFlow-Keras was utilized with the following augmentation techniques:

- Horizontal and vertical flips
- Rotations up to 20 degrees

- Zooming up to 20%
- Shifts up to 20% in both horizontal and vertical directions
- Shearing transformations up to 10%
- Nearest neighbor fill mode for newly created pixels

These mitigate class imbalances and increase the variability of training samples [13], [14].

D. Data Splitting

The data set was initially divided into training and testing sets using a 3:1 ratio. The training set was further divided into training and validation subsets in an 80:20 ratio. This stratification ensures reliable performance monitoring and an unbiased evaluation of model performance during the training and testing phases.

E. Data Analysis and Visualization

Statistical analysis of the data set revealed varying aspect ratios, dimensions, and pixel intensity distributions between images. The mean and standard deviation of the pixel intensities were computed to inform pre-processing standardization. Visualization of random samples from each class demonstrated the diversity of the dataset, while class distribution analysis highlighted existing imbalances, guiding enhancement strategies.

F. Observations

Uniform resizing and normalization effectively minimized preprocessing variability, contributing to consistent model inputs. Extensive data augmentation enhanced the model's robustness and ability to generalize to unseen data. The careful division of data into training, validation, and testing sets ensured a clear and reproducible evaluation framework [9], [13].

Note: The extensive preparation and pre-processing steps outlined above establish a solid foundation for effective model training and evaluation.

IV. Environment Setup

This section details the experimental environment established to evaluate and compare the performance of 15 pretrained models for character-level static hand gesture recognition using the *Sign Language Gesture Images Dataset* [13]. The setup encompasses hardware and software configurations, dataset preparation, model configuration, training parameters, data splitting strategies, and visualization tools, ensuring a comprehensive and reproducible framework for the study.

A. Hardware Configuration

Experiments were executed on the Kaggle platform, leveraging its cloud-based computational resources to ensure consistency and scalability. The specific hardware specifications included:

- **CPU**: Kaggle general-purpose CPU
- **GPU**: NVIDIA Tesla P100 with 16 GB memory, optimized for deep learning tasks
- **RAM**: 32 GB
- **Storage**: 57.6 GB

The NVIDIA Tesla P100 GPU significantly accelerated model training by optimizing matrix computations and enabling the use of larger batch sizes, while the 32 GB RAM facilitated efficient data handling and preprocessing.

B. Software Configuration

The experiments were conducted within Kaggle's Linux-based environment, utilizing the following software stack:

- **Programming Language**: Python 3
- **Deep Learning Frameworks**: TensorFlow, Keras, PyTorch
- **Libraries**: OpenCV, Scikit-learn, Matplotlib, Seaborn, tqdm

All necessary libraries and frameworks were pre-installed, ensuring compatibility and minimizing setup time. The use of both TensorFlow-Keras and PyTorch facilitated the integration of pretrained models not natively supported in a single framework [9].

C. Dataset Preparation

The *Sign Language Gesture Images Dataset* [13] comprises 55,500 images across 37 distinct gesture classes, including alphanumeric labels. The dataset structure is organized into subdirectories, each representing a specific gesture class with images stored in `.jpg` format. Key preprocessing steps included:

- **Image Loading**: Utilized `cv2.imread` to load images, ensuring each image adhered to the shape $(50, 50, 3)$.
- **Normalization**: Pixel values were scaled to the range [0, 1] by dividing by 255.0 to standardize inputs for model training.
- **Label Mapping**: Established direct and reverse class-to-index mappings to facilitate numerical indexing.
- **Data Augmentation**: Applied transformations such as flips, rotations, zooms, shifts, and shearing using TensorFlow-Keras' `ImageDataGenerator` to enhance dataset diversity and model generalization.
- **Data Splitting**: Implemented a 60%-20%-20% split for training, validation, and testing sets, respectively, ensuring balanced class distributions through stratified sampling.

This structured preparation ensured high-quality input data, essential for effective model training and evaluation [9], [13].

D. Model Configuration

A diverse set of 15 pretrained models was selected to provide a comprehensive analysis of architectural effectiveness in static hand gesture recognition. The models included EfficientNetV2L, VGG16, ResNet50, among others, all initialized with ImageNet-trained weights. Key configuration steps involved:

- **Input Dimensions**: Standardized to $(50, 50, 3)$ across all models.
- **Layer Freezing**: Initially froze convolutional layers to retain pretrained feature extraction capabilities.

- **Custom Layers**: Added Global Average Pooling, a Dense layer with 128 neurons and ReLU activation, and a final Dense layer with 37 neurons and softmax activation to adapt models to the specific classification task.
- **Fine-Tuning**: Unfroze final layers to allow models to learn task-specific features during training.

This configuration balanced feature reuse from pretrained models with the adaptation to the specific gesture recognition task [7], [9].

E. Training Parameters

Models were trained under uniform conditions to ensure fair comparison:

- **Optimizer**: Adam, chosen for its adaptive learning rate capabilities [4], [5].
- **Loss Function**: Categorical crossentropy, suitable for multi-class classification.
- **Batch Size**: 32, balancing computational efficiency and memory usage.
- **Epochs**: 20, providing sufficient iterations for learning while mitigating overfitting risks.
- **Learning Rate Strategy**: Adaptive learning rates facilitated by the Adam optimizer to ensure stable and efficient convergence.

These parameters were selected to optimize training efficiency and model performance across all architectures.

F. Data Splitting Strategy

A structured data splitting approach was employed to ensure unbiased evaluation:

- **Training Set**: 60% of the dataset, providing ample data for model learning.
- **Validation Set**: 20%, derived from the training set to monitor performance and guide hyperparameter tuning.
- **Test Set**: 20%, reserved for unbiased final evaluation of model generalization.
- **Stratification and Shuffling**: Ensured balanced class distributions and introduced randomness to eliminate ordering biases.

This strategy facilitated robust training, reliable performance monitoring, and unbiased final evaluations [9], [13].

G. Experiment Execution

The experimental workflow encompassed the following steps:

1) **Data Loading and Preprocessing**: Loaded and normalized images, applied data augmentation during training to enhance generalization.
2) **Model Initialization**: Initialized each pretrained model with ImageNet weights, added custom layers, and configured for the specific classification task.
3) **Training and Validation**: Trained models using the defined parameters, monitored validation performance to prevent overfitting, and saved the best-performing weights based on validation accuracy.
4) **Evaluation**: Assessed models on the test set using accuracy, precision, recall, and F1-score metrics.

This systematic approach ensured consistent and reproducible training and evaluation across all models [9], [13].

H. Visualization Tools

Comprehensive visualization techniques were employed to interpret and compare model performances:

- **Matplotlib and Seaborn**: Generated accuracy and loss curves, confusion matrices, and performance graphs to visualize quantitative metrics.
- **t-SNE**: Reduced high-dimensional feature spaces to 2D for visualizing class separations and clustering behaviors.
- **Grad-CAM**: Produced heatmaps to identify influential regions in input images contributing to model predictions, enhancing interpretability.
- **LIME**: Provided local explanations by highlighting specific pixels or regions impacting individual predictions.

These tools facilitated both quantitative validation and qualitative assessment of model behaviors, offering deeper insights into feature extraction and classification mechanisms [12], [14].

I. Limitations

Despite the robust setup, certain limitations were encountered:

- **Memory Constraints**: Limited RAM posed challenges when training more complex models or handling larger datasets.
- **Framework Compatibility**: Some pretrained models required adaptation to PyTorch due to limited support in TensorFlow-Keras, necessitating modifications in the experimental workflow.

These challenges were mitigated by optimizing resource usage and ensuring uniform experimental conditions across all models [9].

J. Significance of the Setup

The meticulously designed environment ensured that performance variations among models were attributable to architectural differences rather than external factors. By standardizing hardware and software configurations, employing consistent data preprocessing and splitting strategies, and utilizing comprehensive evaluation metrics and visualization tools, the study achieved reliable and reproducible comparisons of pretrained models for static hand gesture recognition [7], [9], [13].

V. Methodology

The experimental approach used to assess and contrast the performance of 15 pretrained deep learning models for static hand gesture detection is described in this section. The methodology ensures a thorough and repeatable framework for the study by including dataset preparation, model selection and configuration, training procedures, evaluation measures, and interpretability techniques.

A. Data Acquisition and Preparation

1) Dataset Description: The study utilizes the *Sign Language Gesture Images Dataset* [13], comprising 55,500 RGB images distributed uniformly across 37 gesture classes, including alphanumeric labels (A-Z, 0-9) and a space gesture represented as "_". Each class contains 1,500 images, ensuring a balanced class distribution. Images are stored in `.jpg` format with dimensions of $50 \times 50 \times 3$ pixels, facilitating computational efficiency in deep learning tasks.

2) Preprocessing Steps: To ensure consistency and compatibility with pretrained models, the dataset underwent several preprocessing steps:

- **Image Loading**: For efficient manipulation images were loaded using OpenCV's `cv2.imread` function and converted to NumPy arrays.
- **Image Resizing**: All images were resized to a uniform dimension of 50×50 pixels with 3 color channels, which aligned with the required inputs of the selected models.
- **Normalization**: Pixel intensity values were initially in the range [0, 255] which were divided by 255.0 and scaled to [0, 1]. This normalization increases numerical stability and accelerates convergence during training.
- **Label Encoding**: Gesture class labels were mapped to unique numerical indices for facilitating efficient class identification by the models using a label mapping dictionary.

3) Data Augmentation: Data augmentation was applied to the training set using TensorFlow's `ImageDataGenerator` to enhance the generalizability of the models and mitigate overfitting. The augmentation techniques included:

- **Horizontal Flip**: Random horizontal flipping of images.
- **Vertical Flip**: Random vertical flipping to simulate orientation variations.
- **Rotation**: Random rotations up to $\pm 20°$ to introduce angular diversity.
- **Zoom**: Random zooming between $0.8\times$ and $1.2\times$ to vary image scale.
- **Shifting**: Translations of up to 20% of the image's width and height in both directions.
- **Shearing**: Shear transformations up to 10% to introduce geometric distortions.
- **Fill Mode**: Nearest neighbor interpolation was used to fill newly created pixels during transformations.

4) Data Splitting Strategy: To ensure balanced class distributions across subsets, the dataset was divided into training, validation, and test sets using a stratified sampling technique:

- **Training Set**: 60% of the data (33,000 images) for model learning.
- **Validation Set**: 20% of the data (11,000 images) for monitoring performance and guiding hyperparameter tuning.
- **Test Set**: 20% of the data (11,000 images) for unbiased evaluation of model generalization.

Reproducibility was ensured by employing a fixed random seed , and shuffling was performed before splitting to eliminate ordering biases.

B. Pretrained Models and Architecture Selection

1) List of Pretrained Models: The study evaluated 15 pretrained models from diverse architectural families to ensure a comprehensive analysis:

- **EfficientNet Variants**: EfficientNetV2L, EfficientNetB0, EfficientNetV2S
- **VGG Series**: VGG16, VGG19
- **ResNet Variants**: ResNet50, ResNet101, ResNet152
- **MobileNet Variants**: MobileNetV2, MobileNetV3Large
- **DenseNet Variants**: DenseNet121, DenseNet169, DenseNet201
- **ConvNeXt Variants**: ConvNeXtBase, ConvNeXt

2) Rationale for Model Selection: These models were chosen due to their efficiency, variety of architectures, and track record of performance in picture categorisation tasks:

- **Architectural Diversity**: By incorporating models with varied depths, widths, and connectivity (for example, DenseNet's dense connections, ResNet's skip connections), it is possible to evaluate various feature extraction capabilities.
- **Computational Efficiency**: Because they are computationally efficient, models such as EfficientNet and MobileNet can be used in contexts with limited resources.
- **Proven Performance**: On benchmark datasets such as ImageNet, these models have demonstrated outstanding performance, which offers a solid basis for transfer learning.

3) Transfer Learning with ImageNet Weights: Weights pretrained on the Ima-geNet dataset were used to initialise all chosen models [7]. With less domain-specific data, models may more successfully adjust to the static hand gesture recognition challenge thanks to transfer learning, which makes use of the generalised feature representations acquired from ImageNet. This strategy has a number of benefits:

- **Feature Generalization**: Pretrained weights capture essential visual features such as edges, textures, and shapes, which are transferable to the gesture recognition domain.
- **Faster Convergence**: Models initialized with pretrained weights require fewer training epochs to achieve optimal performance.
- **Improved Accuracy**: Leveraging transfer learning enhances model performance, especially when the target dataset is limited in size.

C. Model Configuration

1) Frozen Layers for Feature Reuse: To make the most of pretrained feature extraction, the early convolutional layers of each model were frozen during the initial training phase:

- **Preservation of Learned Features**: Freezing these layers helps retain the general patterns learned from ImageNet, ensuring that the model benefits from existing knowledge.
- **Reduced Overfitting**: By reducing the number of trainable parameters, the risk of overfitting on the smaller target dataset is minimized.

2) Adding Custom Layers: To adapt the pretrained models for the specific task of static hand gesture recognition, custom layers were added to each architecture:

- **Global Average Pooling (GAP) Layer**: This layer condenses feature maps into a single vector per image. It lowers the number of trainable parameters, making the model more efficient and reducing overfitting.
- **Dense Layer with 128 Neurons and ReLU Activation**: Introduces non-linearity which allows the model to pick up more complex patterns.
- **Final Dense Layer with 37 Neurons and Softmax Activation**: Corresponds to the 37 hand gesture classes, enabling the model to output class probabilities for multi-class classification.

3) Justification for Design Choices: The incorporation of these custom layers strikes a balance between using pretrained features and allowing the models to learn domain-specific representations:

- **GAP Layer**: Bridges the gap between convolutional and dense layers by summarizing spatial information efficiently.
- **ReLU Activation**: Offers computational simplicity and mitigates the vanishing gradient problem, which helps faster and more stable training [5].
- **Softmax Activation**: Provides a probabilistic interpretation of class predictions, making it ideal for multi-class classification.

D. Training Process

1) Training Parameters: Same set of training parameters were applied across all models to ensure a fair comparison:

- **Optimizer**: Adam optimizer [4], [5], chosen for its adaptive learning rate capabilities and efficient convergence properties.
- **Learning Rate**: A default learning rate of 0.001 was used, suitable for fine-tuning pre-trained models.
- **Batch Size**: Set to 32, balancing memory usage and computational performance.
- **Number of Epochs**: Each model was trained for 20 epochs, providing sufficient iterations for learning while avoiding overfitting.
- **Loss Function**: Categorical crossentropy was used to measure how far predicted probabilities were from the actual class labels, making it a suitable choice for multi-class classification.

2) Training Strategy: To optimize model performance and reduce overfitting, the following strategies were employed:

- **Layer Freezing and Unfreezing**: Initially, convolutional layers were frozen to leverage pretrained features. Later, the final layers were unfrozen to allow the model to adapt to the specific gesture recognition task.
- **Early Stopping and Checkpointing**: Training was monitored, and checkpoints were saved based on validation accuracy. If the model stopped improving, training was halted early to prevent overfitting.

3) Hardware Utilization: Training was performed on Kaggle's NVIDIA Tesla P100 GPU with 16 GB memory, which significantly accelerated computations and allowed for larger batch sizes. The 32 GB RAM supported efficient data handling and preprocessing without significant bottlenecks.

E. Evaluation Metrics

To thoroughly assess model performance, the following metrics were utilized:

1) Accuracy: Accuracy is defined as the ratio of correctly predicted instances to the total number of instances:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

It gives a general idea of how well the model performs but may not be reliable if some classes have significantly more data than others.

2) Precision, Recall, and F1-Score: For a detailed evaluation, especially in multi-class settings, precision, recall, and F1-score were also calculated:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

where TP denotes true positives, FP false positives, and FN false negatives for each class.

Importance of Metrics: While accuracy provides an overall performance snapshot, precision and recall offer insights into the model's ability to correctly identify specific classes and minimize errors, respectively. The F1-score balances these aspects, providing a single metric that accounts for both precision and recall, which is particularly useful in multi-class classification scenarios [9].

F. Visualization and Interpretability Tools

To augment the quantitative evaluation, several visualization and interpretability tools were used to gain deeper insights into model behavior:

1) Confusion Matrices: **Description**: A confusion matrix is a table that compares predicted class labels against true labels, highlighting correct classifications along the diagonal and misclassifications off-diagonal.

Purpose: This breakdown helps pinpoint which classes the model struggles with and reveals any patterns in misclassification.

2) t-SNE (t-Distributed Stochastic Neighbor Embedding): **Description**: t-SNE is a dimensionality reduction technique that projects high-dimensional feature representations into a 2D space, making it easier to visualize.

Purpose: It shows how well the model groups different gesture classes based on learned features. If the classes are well-separated, it means the model can distinguish them effectively. If they overlap, it suggests potential confusion between certain gestures.

3) Grad-CAM (Gradient-weighted Class Activation Mapping): **Description**: Grad-CAM generates heatmaps that highlight the most influential regions of an input image that contributed to the model's predictions.

Purpose: This makes it easier to see whether the model is focusing on the right features. If it's looking at irrelevant areas, adjustments may be needed to improve accuracy.

4) LIME (Local Interpretable Model-Agnostic Explanations): **Description**: LIME explains individual predictions by slightly altering input data and analyzing the impact on how the model's output changes. This highlights influential pixels or regions.

Purpose: It provides instance-specific explanations, helping to verify that model predictions align with meaningful visual cues rather than random patterns.

Significance: Together, these tools enable both global and local analysis of model behavior. Which facilitates a deeper understanding of feature extraction, decision-making processes, and areas for potential improvement [12], [14].

G. Implementation Workflow

The implementation followed a structured and reproducible workflow:

1) **Load and Preprocess the Dataset**: Images were loaded, resized, normalized. Labels were encoded, data augmentation was applied to the training set to enhance variability and improve generalization.
2) **Split the Data**: The dataset was divided into training (60%), validation (20%), and test (20%) sets. Stratified sampling was used to maintain class balance across the splits.
3) **Configure Pretrained Models**: Each of the 15 pretrained models was initialized with ImageNet weights. The early layers were frozen to retain learned features, while custom layers were added to adapt the model for the 37-class classification task.
4) **Train the Models**: Models were trained using the Adam optimizer, categorical crossentropy loss, a batch size of 32, and for 20 epochs. Validation performance was monitored to guide the training process and prevent overfitting.
5) **Evaluate Models**: Trained models were evaluated on the test set using accuracy, precision, recall, and F1-score metrics to assess their overall and class-specific performance.
6) **Analyze Results**: Visualization tools such as confusion matrices, t-SNE plots, Grad-CAM heatmaps, and LIME explanations were used to interpret and compare model behaviors and decision-making processes.

This structured approach ensured consistency and fairness in evaluating all pretrained models, allowing for reliable comparisons and comprehensive analysis of their effectiveness in static hand gesture recognition [9], [13].

H. Significance of the Methodology

The meticulously designed methodology ensures that performance variations among pretrained models are attributable to architectural differences rather than external factors. By standardizing data preprocessing, model configuration, training procedures, and evaluation metrics, the study achieves a fair and unbiased comparison of model capabilities. The integration of interpretability tools further enhances the understanding of model behaviors, making the findings more reliable and meaningful [7], [9], [13].

VI. Performance Analysis

This study analyzes 15 pretrained ImageNet models for character-level static hand gesture recognition. Additionally, this also highlights their strengths and limitations in terms of accuracy, computational efficiency, and robustness. The outcomes aim to guide model selection for gesture-based Human-Computer Interaction (HCI) systems by evaluating the models' performance across several metrics such as Top-1, Top-2, and Top-3 accuracy, precision, recall, and F1-score.

A. General Observations

Table I summarizes the outcomes for all 15 models, offering a wide range of architectural choices. It includes lightweight models like MobileNetV2 [19] and deeper, feature-rich models like ResNet152 [29]. The selection of architecture has a major impact on the trade-off between computational complexity and classification performance.

The ConvNeXtXLarge model [22] achieved the highest Top-1 accuracy of 99.63%, leveraging its modern architectural innovations inspired by Vision Transformers. However, its high parameter count (350.1M) and size (1310 MB) make it unsuitable for deployment in resource-constrained environments.

On the other hand, models such as MobileNetV2 [19] and EfficientNetB0 [18] demonstrated excellent performance with minimal computational cost, making them ideal for edge devices. MobileNetV2 achieved a Top-1 accuracy of 65.49%, while EfficientNetB0 reached 98.67%, demonstrating the trade-off between accuracy and efficiency.

B. Accuracy and Ranking Trends

The VGG series models, VGG16 [16] and VGG19 [17], exhibited remarkable performance, achieving Top-1 accuracies of 98.70% and 99.25%, respectively. Despite their older architectures, they remain competitive due to their depth and dense connectivity.

EfficientNetV2L [15], a well-balanced design achieved a Top-1 accuracy of 95.75

The DenseNet family, specifically DenseNet121 [26] and DenseNet169 [25], efficiently reused features through dense connections and achieved Top-1 accuracies of 89.98

TABLE I
PERFORMANCE METRICS OF SELECTED MODELS

Model	Parameters	Size (MB)	Depth	Epoch	Top-1 Acc. (%)	Top-2 Acc. (%)	Top-3 Acc. (%)	Precision	Recall	F1 Score
EfficientNetV2L	119 M	479.0	-	20	95.75	98.72	99.47	0.96	0.96	0.96
VGG16	138.4 M	528.0	16	20	98.70	99.92	99.99	0.99	0.99	0.99
VGG19	143.7 M	549.0	19	20	99.25	99.99	100.00	0.99	0.99	0.99
EfficientNetB0	5.3 M	29.0	132	20	98.67	99.91	100.00	0.99	0.99	0.99
MobileNetV2	3.5 M	14.0	105	20	65.49	81.65	88.91	0.70	0.65	0.65
DenseNet201	20.2 M	80.0	402	20	90.21	96.73	98.40	0.92	0.90	0.89
MobileNetV3Large	5.4 M	-	-	20	95.93	99.00	99.85	0.97	0.96	0.96
ConvNeXtXLarge	350.1 M	1310.0	-	20	99.63	99.99	100.00	0.99	0.99	0.99
ConvNeXtBase	-	-	-	20	98.98	99.99	100.00	0.99	0.99	0.99
EfficientNetV2S	21.6 M	88.0	-	20	99.24	100.00	100.00	0.99	0.99	0.99
DenseNet169	14.3 M	57.0	338	20	84.33	95.57	98.54	0.88	0.84	0.83
DenseNet121	8.1 M	33.0	242	20	89.98	97.41	99.09	0.92	0.90	0.90
ResNet101	44.7 M	171.0	209	20	95.50	99.63	99.92	0.97	0.95	0.96
ResNet152	60.4 M	232.0	311	20	97.06	99.61	99.88	0.97	0.97	0.97
ResNet50	25.6 M	98.0	107	20	97.28	99.86	99.95	0.98	0.97	0.97

C. Precision, Recall, and F1-Score Analysis

While accuracy provides an overall view of performance, metrics such as precision, recall, and F1-score are crucial for understanding class-wise behavior:

1. **Precision**: Models such as ResNet50 [27] and ConvNeXtBase [24] achieved high precision, illustrating their reliability in avoiding false positives. For example, ResNet50 scored a precision of 0.98, making it perfect for applications that requires high confidence in predictions.

2. **Recall**: DenseNet121 [26] and EfficientNetV2S [23] indicated strong recall values, ensuring minimal false negatives. DenseNet121, with a recall of 0.90, performed well in identifying underrepresented gestures.

3. **F1-Score**: The F1-scores for models like ConvNeXtXLarge [22] (0.99) and ResNet152 [29] (0.97) highlight their balanced performance in precision and recall, making them robust choices for multi-class classification.

D. Resource Efficiency and Deployment Considerations

For deployment in real-world applications, computational efficiency and resource constraints are the key factors. Lightweight models such as MobileNetV2 [19] and MobileNetV3Large [21] provide a substantial reduction in size and parameters, with MobileNetV2 requiring only 14 MB of memory. However, their lower accuracy scores indicate limitations in handling complex gestures.

EfficientNet architectures, particularly EfficientNetB0 [18] and EfficientNetV2S [23], strike a balance between accuracy and efficiency, making them suitable for both edge devices and server-based implementations. EfficientNetV2S achieved a Top-1 accuracy of 99.24% while maintaining a manageable parameter size of 21.6M

E. Visual Insights and Interpretability

Visualization tools such as Grad-CAM and LIME provided qualitative data about model behavior and so validating their decision-making processes. For instance: - Grad-CAM heatmaps highlighted regions of the hand gesture that influenced predictions, ensuring models focus on relevant features. - t-SNE plots demonstrated effective clustering of gesture classes, particularly for models like ConvNeXtXLarge [22], which demonstrated distinct class divisions.

F. Challenges and Future Directions

Although the pretrained models were quite successful, challenges were faced such as intra-class variability, background noise, and occlusion success. Models like ResNet50 and DenseNet201 had trouble with certain gesture classes where significant overlap existed. It is indicating the need for further optimization.

Future research could explore hybrid architectures that combines CNNs with Graph Neural Networks (GNNs) or leveraging Vision Transformers for improved spatial understanding. Furthermore, incorporating advanced data augmentation techniques and larger gesture datasets could improve robustness.

G. Conclusion

The comparative analysis highlights ConvNeXtXLarge [22] as the most accurate model. On the other hand, EfficientNetB0 [18] and MobileNetV2 [19] stand out for resource-constrained scenarios. The results give a basis for selecting pretrained models based on application requirements, contributing to the advancement of gesture-based HCI systems.

ACKNOWLEDGMENT

The authors would like to thank the Kaggle platform for providing the computational resources necessary to conduct this study.

REFERENCES

[1] F. A. Raheem and A. A. Abdulwahhab, “Deep learning convolution neural networks analysis and comparative study for static alphabet ASL hand gesture recognition,” *Journal of Xidian University*, vol. 14, no. 4, pp. 1871–1881, 2020.

[2] K. Myagila and H. Kilavo, “A comparative study on performance of SVM and CNN in Tanzania sign language translation using image recognition,” *Applied Artificial Intelligence*, vol. 36, no. 1, pp. 2005297, 2022.

[3] A. Sultan, W. Makram, M. Kayed, and A. A. Ali, “Sign language identification and recognition: A comparative study,” *Open Computer Science*, vol. 12, no. 1, pp. 191–210, 2022.

[4] D. Hunter, H. Yu, M. S. Pukish III, J. Kolbusz, and B. M. Wilamowski, “Selection of proper neural network sizes and architectures—A comparative study,” *IEEE Transactions on Industrial Informatics*, vol. 8, no. 2, pp. 228–240, 2012.

[5] J. He, L. Li, J. Xu, and C. Zheng, "ReLU deep neural networks and linear finite elements," *arXiv preprint arXiv:1807.03973*, 2018.

[6] D. Grattarola and C. Alippi, "Graph neural networks in TensorFlow and Keras with Spektral [application notes]," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 99–106, 2021.

[7] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[8] A. A. Barbhuiya, R. K. Karsh, and R. Jain, "CNN based feature extraction and classification for sign language," *Multimedia Tools and Applications*, vol. 80, no. 2, pp. 3051–3069, 2021.

[9] O. K. Oyedotun and A. Khashman, "Deep learning in vision-based static hand gesture recognition," *Neural Computing and Applications*, vol. 28, no. 12, pp. 3941–3951, 2017.

[10] M. Huenerfauth, M. Marcus, and M. Palmer, "Generating American Sign Language classifier predicates for English-to-ASL machine translation," Ph.D. dissertation, University of Pennsylvania, 2006.

[11] M. Al-Qurishi, T. Khalid, and R. Souissi, "Deep learning for sign language recognition: Current techniques, benchmarks, and open issues," *IEEE Access*, vol. 9, pp. 126917–126951, 2021.

[12] Z. Lv, F. Poiesi, Q. Dong, J. Lloret, and H. Song, "Deep learning for intelligent human–computer interaction," *Applied Sciences*, vol. 12, no. 22, pp. 11457, 2022.

[13] A. Khan, "Sign Language Gesture Images Dataset," Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/ahmedkhanak1995/sign-language-gesture-images-dataset.

[14] A. Sakhawat, "Sign Language Gesture Images Dataset Description," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/sakhadib/sign-language-gesture-images-dataset-description.

[15] A. Sakhawat, "Character Sign Language Detection using EfficientNetV2L," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/sakhadib/charr-sign-language-detection-efficientnetv2l.

[16] A. Sakhawat, "Character Sign Language Detection using VGG16," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/sakhadib/character-sign-language-detection-vgg16.

[17] A. Sakhawat, "Character Sign Language Detection using VGG19," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/sakhadib/character-sign-language-detection-vgg19.

[18] H. Rahman, "Character Sign Language Detection using EfficientNetB0," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/mdhralif/character-sign-language-detection-efficientnetb0.

[19] H. Rahman, "Character Sign Language Detection using MobileNetV2," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/mdhralif/character-sign-language-detection-mobilenetv2.

[20] H. Rahman, "Character Sign Language Detection using DenseNet201," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/mdhralif/character-sign-language-detection-densenet201.

[21] M. Abedin, "Character Sign Language Detection using MobileNetV3Large," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/minhajulbhuiyan/character-sign-language-detection-mobilenetv3large.

[22] M. Abedin, "Character Sign Language Detection using ConvNeXtXLarge," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/minhajulbhuiyan/character-sign-language-detection-convnextxlarge.

[23] M. Abedin, "Character Sign Language Detection using ConvNeXtBase," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/minhajulbhuiyan/character-sign-language-detection-convnextbase.

[24] N. Islam, "Character Sign Language Detection using EfficientNetV2," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nabilaislam210042111/character-efficientnetv2-sign-language-detection.

[25] N. Islam, "Character Sign Language Detection using DenseNet169," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nabilaislam210042111/character-densenet169-sign-language-detection.

[26] N. Islam, "Character Sign Language Detection using DenseNet121," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nabilaislam210042111/character-sign-language-detection-densenet121.

[27] N. Tasneem, "Character Sign Language Detection using ResNet101," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nazifatasneem13/character-sign-language-detection-resnet101.

[28] N. Tasneem, "Character Sign Language Detection using ResNet152," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nazifatasneem13/character-sign-language-detection-resnet152.

[29] N. Tasneem, "Character Sign Language Detection using ResNet50," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/code/nazifatasneem13/character-sign-language-detection-resnet50.

[30] Keras Team, "Keras Applications," Keras 3 API Documentation, 2024. [Online]. Available: https://keras.io/api/applications/.