Introduction

This project focuses on adapting various CNN-based architectures to identify the letters in three different sign language fingerspelling datasets:

- **LIS**: Italian Sign Language,
- **ASL**: American Sign Language,
- **BSL**: British Sign Language.

To achieve this, we employed five pre-trained networks and applied fine-tuning techniques to maximize accuracy. Specifically, we:

- **experimented with Batch Sizes and Epochs**: testing and training with different combinations of batch sizes (16, 32, 64) and epochs (10, 15, 20).
- **incorporated batch normalization and dropout**: improving generalization and reducing overfitting;
- **balanced the datasets**: augmenting or reducing the datasets to ensure balance both within individual datasets and across datasets.

Results Storage

The results from each model on each dataset are stored as images in the branches named after the respective models. These images include:

1. **confusion matrices**: visualizing performance for each dataset and batch size/epoch combination (16-10, 32-15, 64-20);
2. **summary tables**: Providing an overview of the model's performance metrics (accuracy, precision, recall, and F1 scores) across all datasets.

These results are referenced throughout this document to highlight key findings and comparisons.

Datasets

The datasets used for this project were sourced from Kaggle; while some of the datasets originally included samples for numbers, only the folders containing letters were considered. Below is an overview of the datasets:

- **ASL Dataset**:
    - the largest dataset with 26 classes (A-Z);
    - each class contains 3,000 images;
- **LIS Dataset**:
    - the smallest and most unbalanced dataset with 22 classes;
    - class sizes range from 240 to 330 images;
- **BSL Dataset**:
    - the most distinct and noisy dataset with 23 classes;

    - each class contains 1,000 images;
    - some images were blurred or showed more of the person performing the sign rather than focusing on the hands.

The letters that were not represented in the dataset (H, J, Y for the BSL while G, J, S, Z for the LIS) were omitted by the creators themselves since these signs needed movement to be performed correctly.

The BSL dataset, due to the considerable size of its files, has been modified using `Dataset_Resizing.py`: the files have been converted from .png to .jpg, with their size reduced by 75%.

Due to GitHub limitations on file size, the datasets have been uploaded on a shared Google Drive folder (https://drive.google.com/drive/folders/1t2bYdvcHNPQheHlmNWSSlvmu2JbSp4wc)

Dataset Balancing

To address class imbalances and differences across datasets, we manipulated the data using `Dataset_Balancing.py`. This resulted in six balanced datasets:

- **ASL(1000)**: reduced to 1,000 images per class;
- **ASL(300)**: reduced to 300 images per class;
- **BSL(1000)**: original dataset with no changes;
- **BSL(300)**: reduced to 300 images per class;
- **LIS(300)**: augmented to 300 images per class;
- **LIS(1000)**: augmented to 1,000 images per class.

Combined Dataset

To enable cross-dataset learning, the ASL(300), BSL(300), and LIS(300) datasets were merged using `create_combined_dataset.py`. This resulted in a **Combined Dataset**, containing balanced samples from all three sign language alphabets.

ResNet50

ResNet50 is a deep learning model launched by Microsoft Research. Its 50 layers architecture focuses on residual learning and involves:

- input layer, it accepts (224, 224, 3) tensors (224×224 RGB images)
- convolutional layer, with a 7x7 filter, 64 filters, and a stride of 2, to detect features and generate feature map.
- 3x3 max pooling layer with a stride of 2, to the spatial dimensions, retaining important information while reducing computational load
- 4 residual blocks, each one with a specific number of units of 1x1 convolution, 3x3 convolution, and again 1x1 convolution and a variable number of filters

- an average pooling layer, to reduce the spatial dimensions of the feature maps to a single value per feature, simplifying the architecture
- a fully connected layer with 1000 neurons
- a softmax activation in the output layer, where the reduced feature maps undergo a final classification step producing the model's predictions

The key feature of this model is the use of the residual blocks, that connect the activations of previous layers with the next one. This allows the network to skip intermediate layers in order to mitigate the vanishing/exploding gradient problem, enabling smoother training and faster convergence.

Results

To thoroughly test the capabilities of ResNet50, which has approximately twice the number of trainable layers compared to the other architectures, the model was evaluated on datasets of both sizes: 300 images per class (as the other architectures) and 1000 images per class.

In order to maximize the performances of this model, each one of the 7 datasets has been tested with 6 trials, varying by 3 factors:

- Learning rate: 1e-4 or 1e-5,
- Base model layers unfrozen: none or only BatchNormalization layers:

    > BatchNormalization layers contain learned parameters (mean and variance) that align the model with the original dataset's distribution. By unfreezing these layers, the model can adapt to the new dataset, improving accuracy, especially for large datasets [5])

- Dense regularization: adding L2 regularization to the Dense layer or not:

    > Dense layer regularization, through L2 penalties, discourages large weight magnitudes, which can lead to overfitting [6]

Trials where BatchNormalization layers are unfrozen and Dense regularization with L2 penalty is added generally improve results, especially for large datasets, stabilizing the performance by reducing overfitting.

Trials	Learning Rate	Unfrozen Layers	Dense Regulation
1	0,0001	None	No
2	0,00001	None	No
3	0,00001	BatchNormalization	No
4	0,00001	BatchNormalization	Yes
5	0,0001	BatchNormalization	Yes
6	0,0001	BatchNormalization	No

TRIALS	BATCH-EPOCHS	BSL(1000)	BSL(300)	ASL(1000)	ASL(300)	LIS(1000)	LIS(300)	COMBINED
Trial 1	16-10	Overfitted	87%	Overfitted	96%	89%	96%	93%
	32-15	Overfitted	92%	Overfitted	97%	91%	96%	95%
	64-10	Overfitted	92%	Overfitted	96%	90%	97%	95%
Trial 2	16-10	90%	65%	94%	83%	73%	80%	78%
	32-15	91%	64%	95%	85%	72%	80%	79%
	64-10	90%	65%	94%	84%	71%	81%	77%
Trial 3	16-10	92%	68%	95%	81%	78%	82%	80%
	32-15	94%	67%	96%	84%	79%	83%	82%
	64-10	93%	68%	95%	81%	79%	83%	81%
Trial 4	16-10	93%	69%	95%	80%	78%	82%	78%
	32-15	94%	70%	96%	82%	80%	84%	80%
	64-10	93%	98%	95%	85%	79%	82%	80%
Trial 5	16-10	Overfitted	95%	Overfitted	98%	95%	Overfitted	97%
	32-15	Overfitted	96%	Overfitted	Overfitted	96%	Overfitted	Overfitted
	64-10	Overfitted	96%	Overfitted	Overfitted	Overfitted	Overfitted	Overfitted
Trial 6	16-10	Overfitted	97%	Overfitted	99%	95%	Overfitted	97%
	32-15	Overfitted	97%	Overfitted	Overfitted	96%	Overfitted	Overfitted
	64-10	Overfitted	96%	Overfitted	Overfitted	Overfitted	Overfitted	Overfitted

From this table the following observations can be made:

- The LIS(1000) dataset achieved high accuracy without the need for a lower learning rate. Its sharp contrast between the hand and the clean and uniform black background made feature extraction straightforward. This reduced the risk of overfitting or confusion, even with higher learning rates.

- The LIS(300) dataset, despite being smaller, also performed well due to the clarity and simplicity of the images. The consistent background and isolated gestures allowed the model to generalize effectively.

- The ASL(1000) dataset faced challenges due to lower resolution and poor lighting, leading to low contrast between the hand and background. Using BatchNormalization layers with a lower learning rate allowed the model to adapt slowly to the noise and complexity of the dataset. Regularizing the Dense layer also helped improve generalization by mitigating the impact of noise and lighting variability, particularly for

- The BSL datasets posed the greatest difficulty due to complex backgrounds and some gestures involving full-body poses or contextual elements. The increased complexity of the BSL(1000) input space required slower optimization to prevent the model from focusing on irrelevant features in the background, while for both BSL(1000) and BSL(300) the use of the BatchNormalization layer led to better results

- The results for the Combined dataset, with its integration of samples from multiple datasets, despite their varying complexities, showcased the model's ability to learn and generalize from diverse input spaces

This model achieved almost similar results for each dataset regardless of its size (1000 or 300). Achieved accuracy ranged between 89%-97% across all datasets, with small challenges that arose in distinguishing similar gestures:

- BSL: letters M and N
- ASL: letters R, U and V
- LIS: letters M and N, letters R and U

EfficientNetV2

The EfficientNetV2 network is a CNN architecture based on MobileNetV2.

It employs a scaling method that uniformly adjusts the network's width, depth, and resolution using a set of fixed scaling coefficients.
To achieve better accuracy and efficiency, it is crucial to balance all three dimensions network width, depth, and resolution during scaling.

Compound Scaling Method:

1. Fix φ = 1 and perform a small grid search to determine the values of α, β and γ.
2. Fix α, β and γ as constants, and scale up the baseline network using different values of φ.

The equations used to describe depth, width, and resolution are as follows:

$$depth: d = \alpha^\varphi$$
$$width: w = \beta^\varphi$$
$$resolution: r = \gamma^\varphi$$

The architecture also incorporates **squeeze-and-excitation blocks**, which consist of the following components:

1. **Global Average Pooling Layer**: Reduces the spatial dimensionality of the feature maps.
2. **Fully Connected Module**: A small two-layer fully connected network to model the interdependencies between channels.
3. **Rescaling**: Adjusts (scales) the original feature maps based on the learned channel interdependencies.

Results

The results for EfficientNetV2 on the ASL dataset are available in the **EfficientNetV2** branch within the **ASL_res** folder, in the file **Report_EfficientNet_ASL-300_model**.
Across the three batches, we obtained high performance with no significant differences between them. The worst-performing batch, X, still achieved results above 0.60.

The results for EfficientNetV2 on the BSL dataset are available in the **EfficientNetV2** branch within the **BSL_res** folder, in the file **Report_EfficientNet_BSL-300_model**.
For the three batches, we observed slightly lower results compared to ASL but still very high overall. There was some confusion between the letters *M* and *N*.

The results for EfficientNetV2 on the LIS dataset are available in the **EfficientNetV2** branch within the **LIS_res** folder, in the file **Report_EfficientNet_LIS-300_model**.
For the three batches, the results were high, similar to those for ASL. However, there was a tendency to misclassify *F* as *B* and *C* as *O*.

The results for EfficientNetV2 on the Combined dataset are available in the **EfficientNetV2** branch within the **Combined_res** folder, in the file **Report_EfficientNet_Combined-300_model**.
High performance was also achieved across all three batches for the combined dataset.

MobileNetV2

MobileNet is a CNN architecture based on an inverted residual structure, where residual connections occur between the bottleneck layers.

Architecture of MobileNetV2:

1. **Input**: An input image of size 224×224×3 .
2. **Initial Convolution**: A convolutional layer (*conv2d*) with a stride of 2, producing an output with 32 channels.
3. **Bottleneck Layers**: A series of 19 bottleneck layers. Each bottleneck block is defined by:
    - **Expansion block**: Expands the number of channels to *tk*.
    - **Depthwise convolution block**: Applies a 3×3 kernel to each channel independently.
    - **Linear reduction block**: Reduces the number of channels to *k'*.
    - These blocks are repeated *n* times based on the configuration.
4. **1×1 Convolution**: A pointwise convolution.

5. **Average Pooling**: A 7×7 average pooling layer.
6. **Final Output**: A 1×1 convolution produces the final output with kkk classes.

The **inverted residual structure** enhances the network's ability to propagate gradients efficiently across multiple layers while being more memory-efficient.

Results

The results for MobileNetV2 on the ASL dataset are available in the **EfficientNetV2** branch within the **ASL_res** folder, in the file **Report_MobileNet_ASL-300_model**.
Across the three batches, we obtained acceptable results: most predictions are aligned with the main diagonal, although some letters performed better than others.
The model fails to recognize *I* and shows mediocre performance with *E*, *Y*, *V*, and *X*.

The results for MobileNetV2 on the BSL dataset are available in the **EfficientNetV2** branch within the **BSL_res** folder, in the file **Report_MobileNet_BSL-300_model**.
Across the three batches, the results remained acceptable but were slightly worse than for ASL, with most predictions still aligning with the main diagonal.
The model fails to recognize *L*, *M*, and *N*, and has difficulties identifying *A*, *O*, and *T*.

The results for MobileNetV2 on the LIS dataset are available in the **EfficientNetV2** branch within the **LIS_res** folder, in the file **Report_MobileNet_LIS-300_model**.
Across the three batches, the results were similar to those for BSL, with most predictions aligned with the main diagonal.
The model fails to recognize *D* and *K* and struggles with *F*, *M*, *P*, *Q*, *U*, *V*, and *X*.

The results for MobileNetV2 on the Combined dataset are available in the **EfficientNetV2** branch within the **Combined_res** folder, in the file **Report_MobileNet_Combined-300_model**.
Across the three batches, the results were still similar, with most predictions aligned with the main diagonal.
The model fails to recognize the following letters:

- For the LIS dataset: *X*, *U*, *O*, *I*, *K*, *D*.
- For the BSL dataset: *U*, *T*, *N*, *L*.

VGG16 & VGG19

VGG is a family of deep convolutional neural networks known for its simple architecture; these models vary by depth and number of layers with tunable parameters, offering four variants with 11, 13, 16, and 19 layers. Despite the differences in number of convolutional layers, they share a common structure:

- **input layer**: accepts a colored RGB image, so a 224x224x3 tensor;
- **convolutional layers**: the filters with a small receptive field (3x3) and stride of 1; the resolution is kept after each convolution thanks to row and column padding;
- **max pooling layers**: use a 2x2 kernel with a stride of 2 to reduce spatial dimensions;
- **fully connected layers**: 3 layers to capture complex patterns;

- **output layer**: a softmax activation function for classification.

The hidden layers use the ReLU activation function to introduce non-linearity and enhance feature extraction.

The main differences between VGG16 and VGG19 are:

- **VGG16** contains 13 convolutional layers and 3 fully connected layers;
- **VGG19** adds 3 more convolutional layers, making it deeper and capable of learning more complex features. However, this additional depth increases the need for more training data and computational resources.

For the experiments, VGG19 was expected to perform better, particularly for the 32/15 batch size and epoch configuration since this combination provide a a good balance between training and generalization: more epochs allow the model to learn effectively, while a smaller batch size helps capture finer details, avoiding overly smooth gradients.

Results

1. **LIS(300)**:
    - the simpler architecture of VGG16 proved sufficient to model the data, with VGG19 showing no significant improvement;
2. **BSL(300)**:
    - the best results were achieved for the 32/15 configuration, as expected. VGG19 successfully resolved a common issue with VGG16: misclassification of the letter "C.";
3. **ASL(300)**:
    - training with fewer epochs (16/10 configuration) led to underfitting for VGG19, whereas optimal results were observed with the 32/15 configuration;
4. **Combined:**
    - unexpectedly, VGG16 performed better with the combined dataset, whereas VGG19 exhibited poorer results, possibly due to the increased complexity of the data.

Detailed evaluation metrics, including accuracy, precision, recall, F1 scores and confusion matrices, for each dataset and batch/epoch configuration are provided in the corresponding tables in the VGG16 and VGG19 branches.

Final results

The analysis of model performance across the individual and combined datasets (summed up in the following tables) reveals the following key points:

ResNet50 emerged as the top-performing model in terms of accuracy for both the individual datasets and the combined dataset. This highlights its ability to generalize across varied data distributions effectively;

for **individual datasets**, **VGG19** consistently achieved the second-best performance, leveraging its deeper architecture to learn complex features;

for the **combined dataset**, **EfficientNetV2** too demonstrated strong performance, highlighting its efficiency and scalability for mixed data sources.

Dataset-Specific Observations

LIS (300): VGG19 achieved the best performance (accuracy = 0.98), followed closely by ResNet50 (accuracy = 0.96).

BSL (300): ResNet50 led with an accuracy of 0.97, while VGG19 performed similarly well (accuracy = 0.96).

ASL (300): ResNet50 outperformed the other models with an accuracy of 0.96, followed by VGG19.

Combined Dataset ResNet50 maintained its robustness across different configurations for the combined dataset, whereas other models, like MobileNetV2, struggled with significantly lower performance (accuracy < 0.8).

To sum up

ResNet50 is the recommended model for tasks requiring high accuracy and robustness across diverse datasets.

VGG19 may be preferred for specific individual datasets where deeper feature learning is advantageous.

EfficientNetV2 offers a balanced alternative, particularly for the combined dataset, due to its efficiency and competitive accuracy.

Individual datasets

Batch/ Epochs	Dataset	EfficientNetV2	MobileNetV2	VGG16	VGG19	ResNet50	
16/10	LIS	0.93	0.73	0.97	0.98	0.96	0.89
	BSL	0.95	0.74	0.82	0.92	0.97	0.93
	ASL	0.95	0.78	0.83	0.75	0.96	0.95
32/15	LIS	0.94	0.74	0.97	0.98	0.96	0.91
	BSL	0.88	0.75	0.89	0.96	0.97	0.94
	ASL	0.96	0.76	0.86	0.97	0.97	0.96
64/20	LIS	0.93	0.73	0.97	0.97	0.97	0.90
	BSL	0.90	0.70	0.87	0.93	0.96	0.93
	ASL	0.96	0.77	0.87	0.94	0.96	0.95

Combined datasets

Batch/ Epochs	EfficientNetV2	MobileNetV2	VGG16	VGG19	ResNet50
16/10	0.92	0.74	0.89	0.84	0.93
32/15	0.93	0.75	0.94	0.83	0.95
64/20	0.92	0.75	0.96	0.76	0.95

Works cited

1] Grassknoted. ASL Alphabet. Kaggle.

https://www.kaggle.com/datasets/grassknoted/asl-alphabet

[2] Nisopoli, Nicholas. LIS Dataset. Kaggle.

https://www.kaggle.com/datasets/nicholasnisopoli/lisdataset

[3] Tatepe, Eren. BSL Numbers and Alphabet Hand Position for MediaPipe. Kaggle.

https://www.kaggle.com/datasets/erentatepe/bsl-numbers-and-alphabet-hand-position-for-mediapipe

[4] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[5] He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. arXiv, 2015.

[6] Thakur, Ayush. Keras Dense Layer: How to Use It Correctly. Weights & Biases

[7]Mark Sandler and Andrew Howard and Menglong Zhu and Andrey Zhmoginov and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks." arXiv preprint arXiv:1801.04381 (2019).

[8]Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv preprint arXiv:1905.11946(2019).

[9] Simonyan, Karen. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[10] Amerini I. "Lecture 9: Recognition" Computer Vision course, Università di Roma La Sapienza, PowerPoint presentation (A.A. 2023-2024)