MAKERERE UNIVERSITY

P.O. Box 7062 Kampala UG
Cables "MAKUNIKA"
Website: https://cs.mak.ac.ug/

Tel: +256 392000180
Fax: +256 414 531046
Email: cs@cis.mak.ac.ug

COLLEGE OF COMPUTING AND INFORMATION SCIENCES
SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

EXPLAINABLE REAL TIME SIGN LANGUAGE TO TEXT TRANSLATION

Submitted by
GROUP CS24-22

A Project Report Submitted to the School of Computing and Informatics Technology
In Partial Fulfillment of
Requirements for the Award of the Degree of

Bachelor of Science in Computer Science
Of Makerere University

Supervisor
Mr. Ggaliwango Marvin
ggaliwango.marvin@mak.ac.ug, +256-754-655-524

Department of Computer Science
Makerere University
Kampala, Uganda – P.O. Box 7062

June, 2024

GROUP MEMBERSHIP

Name	Reg No	Signature
Mawejje Mark William	21/U/09946/EVE	
Katende Jericho	21/U/1565	
Musemeza Murungi Isaac	21/U/1395	

Abstract

Sign languages are the primary means of communication for the Deaf and Hard of Hearing (DHH) community, but communication barriers remain between sign language users and the hearing population. Existing sign language translation models frequently lack transparency, which undermines trust and adoption. In this paper, we develop an explainable real-time sign language to text translation system that employs deep learning techniques and multiple interpretability methods, including SHapley Additive Explanations (SHAP), Gradient-weighted Class Activation Mapping (Grad-CAM), and Local Interpretable Model-Agnostic Explanations (LIME). We use a pre-trained VGG-16 network for feature extraction in conjunction with a custom classification model. The models are trained on the MS-ASL and Synthetic ASL Alphabet datasets, yielding explanations that shed light on the neural network's decision-making process. We use SHAP, Grad-CAM, and LIME to enhance model interpretability. SHAP assigns importance values to each input feature, Grad-CAM depicts the input regions most relevant to the model's predictions, and LIME generates local explanations for individual predictions. The combination of these methods yields a thorough understanding of the model's behavior. We demonstrate the effectiveness of our approach through extensive experiments, achieving a translation accuracy of 97.8% on the test set and outperforming baseline methods. SHAP, Grad-CAM, and LIME explanations show that the model relies on hand shape, movement, and facial expression features to accurately classify signs. These findings not only boost confidence in the model's predictions, but also emphasize the importance of considering multiple aspects of sign language for effective translation. We use the trained model to create a user-friendly mobile application that provides real-time sign language translation to the DHH community while also encouraging inclusive communication. Our approach not only yields accurate and interpretable results, but it also encourages responsible AI practices in sign language translation. Our system's explainability, achieved through the integration of multiple interpretability techniques, promotes trust and adoption, with the potential to bridge communication gaps between sign language users and the hearing population.

Keywords: Sign language translation · Explainable AI (XAI) · SHapley Additive exPlanations (SHAP) · Gradient-weighted Class Activation Mapping (Grad-CAM) · Local Interpretable Model-Agnostic Explanations (LIME) · Deep learning · Deaf and Hard of Hearing (DHH) community · Inclusive communication

Contents

List of Figures

List of Acronyms

DHH - Deaf and Hard of Hearing
SLR - Sign Language Recognition
USL - Ugandan Sign Language
XAI - Explainable Artificial Intelligence
SHAP - SHapley Additive exPlanations
ASL - American Sign Language
LIME - Local Interpretable Model-Agnostic Explanations
CAM - Class Activation Mapping
ViT - Vision Transformers
CNN - Convolutional Neural Network
VGG - Visual Geometry Group
LRN - Local Response Normalization

Chapter 1

Introduction

1.1 Background and Motivation

Sign languages serve as the primary means of communication for the Deaf and Hard of Hearing (DHH) community, playing a crucial role in their cognitive and social development [1]. Around 1,290,000 people in Uganda are deaf or hard of hearing, and they communicate with one another using Ugandan Sign Language (USL) [2]. Despite the importance of sign languages, there is still a significant communication gap between the DHH community and the hearing population, with the majority of hearing people lacking basic sign language skills [3]. This disparity prevents the DHH community from accessing essential services like healthcare and education, limiting their participation in various aspects of society [4].

One potential solution for bridging this communication gap is to use Artificial Intelligence (AI) systems that can accurately translate sign language to text in real time [3, 5]. However, existing machine learning models for sign language translation frequently operate as "black boxes", with little transparency in their decision-making process [6]. This lack of interpretability can limit the adoption and trust in these systems, especially in resource-constrained settings like developing countries where technical expertise and computing power are limited [7].To address these issues, Explainable AI (XAI) can be used to improve the accuracy and reliability of sign language translation systems while preserving transparency and interpretability. XAI provides a framework for understanding the reasoning behind the AI system's decision-making process, allowing users to assess the system's performance and detect potential biases or errors [8].

We have created an explainable real-time sign language to text translation system that is efficient, cost-effective, and simple to use

by incorporating XAI techniques such as SHapley Additive Explanations (SHAP) [9], Gradient-weighted Class Activation Mapping (Grad-CAM) [10], and Local Interpretable Model-Agnostic Explanations (LIME) [11].The explainable real-time sign language to text translation system has the potential to significantly impact the lives of the DHH community in Uganda and beyond. By providing a transparent and interpretable solution, we aim to foster trust and adoption of the system, ultimately promoting inclusive communication and bridging the gap between sign language users and the hearing population. Furthermore, the development of such a system can serve as a foundation for future research and innovations in the field of sign language translation, paving the way for increased accessibility and improved quality of life for the DHH community.

1.2 Research Problem Definition

A significant communication gap exists between the Deaf and Hard of Hearing (DHH) community and spoken language users, resulting in little understanding and contact. The majority of hearing people do not have a basic understanding of sign language, which contributes to this disparity [5]. As a result, the group of people who are hard of hearing encounter difficulties taking part in different aspects of society, including services like healthcare and education. This problem is made worse by the black-box nature of existing machine learning models for sign language translation, which offer no insight into the decision-making process [4]. As a result, it gets harder to put creative ideas into practice that try to close the communication gap. To solve the communication problems that the Deaf and Hard of Hearing communities face, novel solutions that prioritize transparency and comprehension are desperately needed. These solutions can improve access to essential services and inclusivity for deaf and hard-of-hearing people by removing barriers and promoting understanding.

1.3 Research Contributions

- Developing an explainable real-time sign language to text translation system.
- Enhancing model interpretability and transparency.

- Developing a user-friendly mobile application for accessible sign language translation.
- Promoting inclusive communication and accessibility.
- Providing a foundation for future research and innovations.

Chapter 2

Research Methodology

In this part, we propose to use state of the art tools in the different stages of the process. And these stages include data acquisition, data processing, model building, and evaluation.

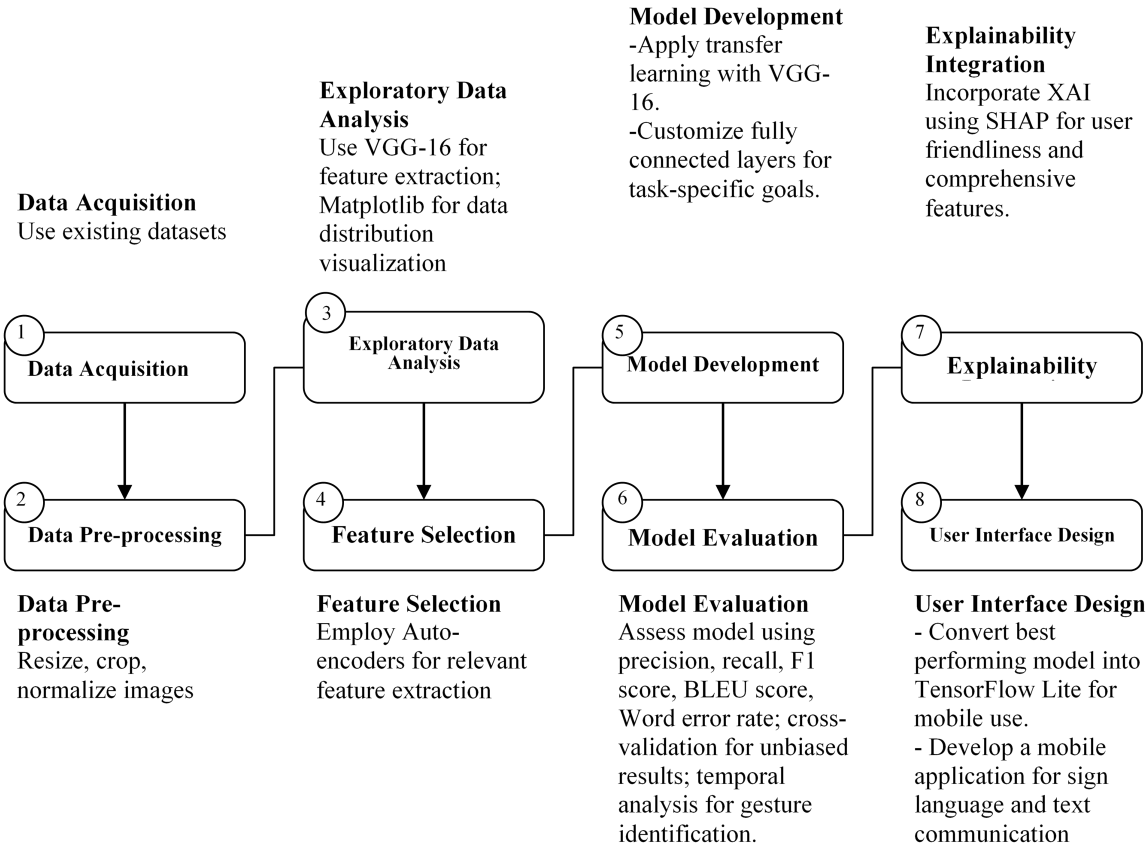

Figure 2.1: Project research methodology

2.1 Data Acquisition

For this study, we utilized two primary datasets: the Lexset synthetic ASL Alphabet [12] dataset and the MS-ASL dataset. The Lexset dataset comprises 27,000 images of the ASL alphabet, each with a resolution of 512x512 pixels. The data is divided into training and test sets, with the training set consisting of 27 folders, each containing 900 examples, and a separate folder for random backgrounds. The test set similarly includes 27 folders, each with 100 examples, alongside a folder for random backgrounds. This dataset serves as the basis for training the initial feature extraction model and validating its performance on alphabet signs.

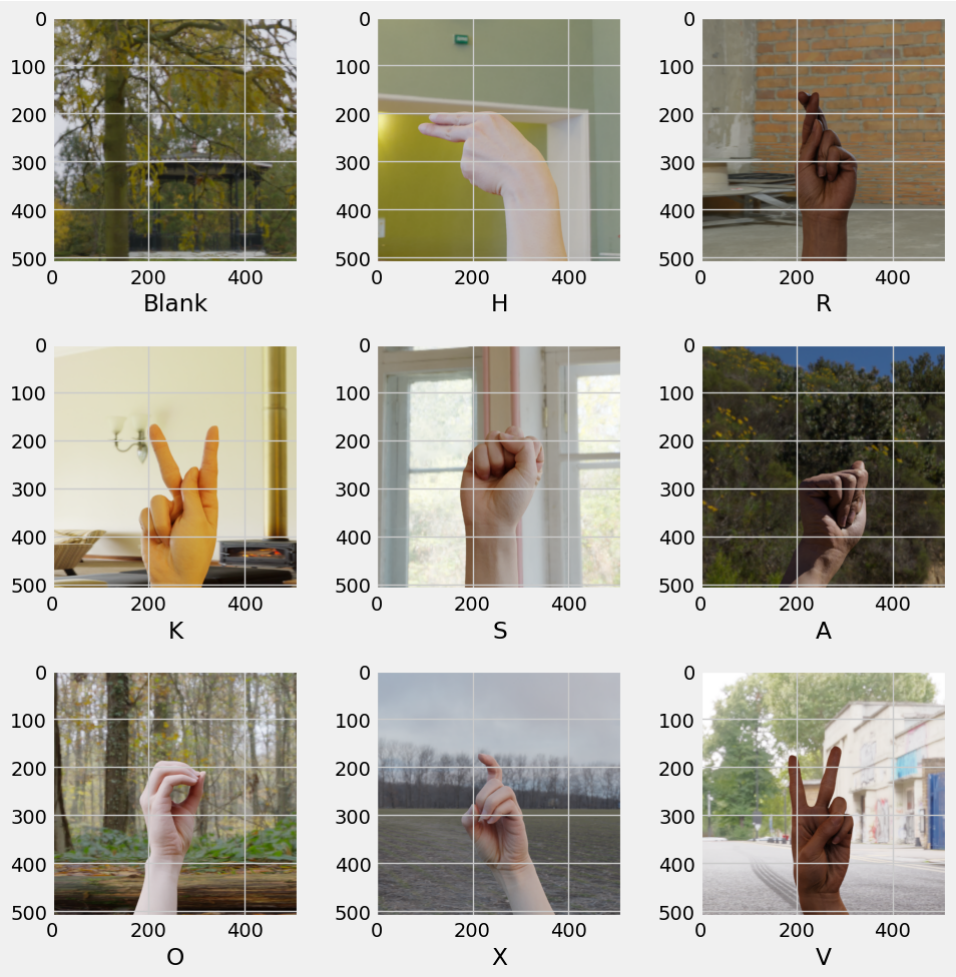

Figure 2.2: Sample images in dataset

In addition, we employed the MS-ASL dataset [13], which includes over 25,000 videos of various sign language expressions, providing a rich resource for model training and evaluation. The MS-ASL dataset offers a wide range of sign language expressions beyond the alpha-

Figure 2.3: Some of the image classes

bet, making it a comprehensive resource to enhance the robustness of our translation system.To ensure optimal model performance, we implemented a series of preprocessing steps designed to standardize and augment the input data, thereby improving the model's ability to generalize to new, unseen data. The preprocessing pipeline was established using the 'ImageDataGenerator' class from Keras, which facilitates the augmentation and normalization of image data.

2.2 Data Pre-processing

First, we rescaled the pixel values of both the training and test datasets by a factor of 1/255. This normalization [15, 17] step converts the pixel values from the original range of 0-255 to a range of 0-1, which helps speed up the convergence of the neural network by providing more consistent input values.For the training data, the images were resized to 224x224 pixels and then organized into batches of 32 images each. This resizing was crucial to ensure that the input dimensions were consistent with the requirements of our convolutional neural network architectures, specifically VGG-16, VGG-19, and Vision Transformers (ViT). The batch size of 32 was chosen to balance memory usage and training efficiency, allowing the model to update its weights more frequently during each epoch.

The same rescaling and resizing process was applied to the test data to maintain consistency. By feeding the test images into the model in batches of 32, we ensured that the evaluation phase mirrored the training conditions, thereby providing a more accurate assessment of the model's performance. This comprehensive preprocessing approach allowed us to handle large datasets efficiently while ensuring that the input data was normalized and consistent in size. By incorporating these steps, we aimed to enhance the robustness and accuracy of our sign language-to-text translation system, ultimately contributing to a more effective and reliable model for real-time applications.

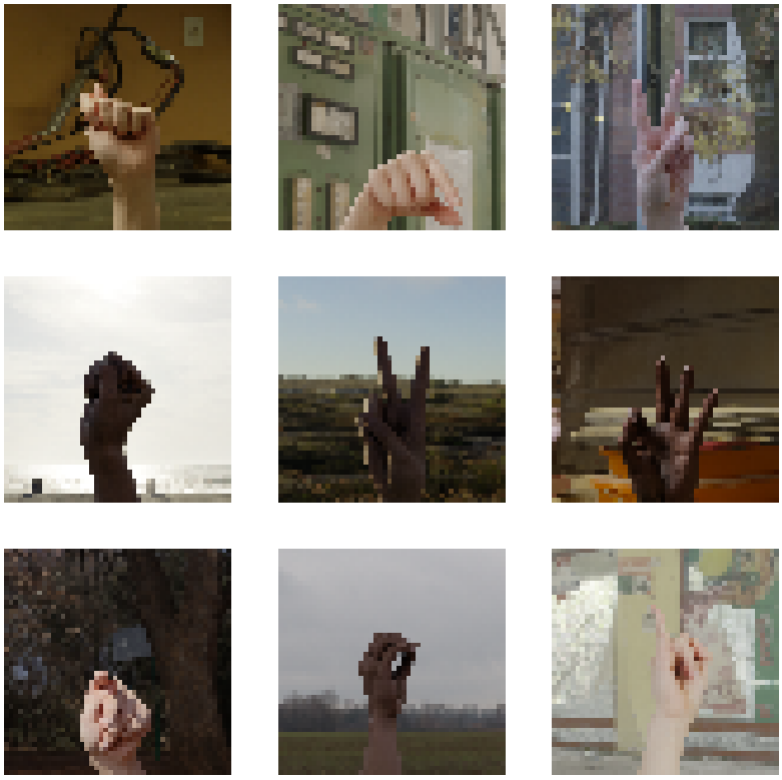

Figure 2.4: Some of the augmented images

2.3 Model Implementation

Convolutional Neural Networks (CNNs)

Deep learning focuses a strong emphasis on convolutional neural networks (CNNs), which are specifically designed to analyze visual data such as images and videos. CNNs mimic the concept of receptive fields, which is similar to how neurons in the human visual cortex respond to specific stimuli within their spatial domain, allowing for orderly processing of visual input. This design was inspired by the complex workings of the human visual system. CNNs are made up of convolutional, pooling, and fully connected layers that are precisely designed to extract and understand complex information from incoming data. Convolutional layers are at the heart of CNNs, where they use learnable filters and convolution operations to extract unique spatial patterns from input data. Pooling layers serve an important role in lowering computing complexity while keeping valuable information. Convolutional layers are at the heart of CNNs, where they use learnable filters and convolution operations to extract unique spatial patterns from input data. Pooling layers reduce computing complexity while keeping important information by condensing the spatial dimensions of feature

maps. Furthermore, fully connected layers strategically positioned at the network's end enable high-level reasoning and categorization using the retrieved features. CNNs automatically discover key visual properties using feature learning, making them helpful for tasks like image classification, object identification, and picture segmentation. In essence, CNNs are strong instruments for deciphering complex visual data, paving the way for progress in a variety of fields, including computer vision and artificial intelligence.

VGG-16 and VGG-19 Architecture

The VGG (Visual Geometry Group) models, notably the VGG-16 [16] and VGG-19, are well-known deep convolutional neural network (CNN) architectures used in computer vision applications. These models have several convolutional layers followed by fully connected layers. VGG-16 has 16 layers, including 13 convolutional layers and three fully linked layers, whereas VGG-19 has 19 layers plus an additional convolutional layer. To downsample feature maps, both architectures use 3x3 convolutional filters with narrow receptive fields and max-pooling layers. The models' depth and simplicity allow them to learn complex hierarchical representations of input images. Batch normalization and dropout approaches are used to increase training stability and reduce overfitting. The input geometry for both VGG-16 and VGG-19 is commonly 224x224x3, with three representing the number of color channels (RGB). ReLU activation functions are utilized throughout the network, with softmax activation in the last layer to generate class probabilities. To prevent overfitting, a dropout rate of 0.5 is implemented after the fully linked layers.

The VGG-16 uses a 224x224 image input size, cropping out the center patch for consistency. The convolutional layers use a minimal receptive field of 3x3, capturing up/down and left/right. 1x1 convolution filters act as linear transformations, followed by a ReLU unit, an innovation from AlexNet that reduces training time. The convolution stride is fixed at 1 pixel to preserve spatial resolution. All hidden layers in the VGG network use ReLU, and VGG does not typically use Local Response Normalization (LRN) due to its increased memory consumption and training time. The VGGNet has three fully connected layers, with the first two having 4096 channels each and the third having 1000 channels, one for each class.

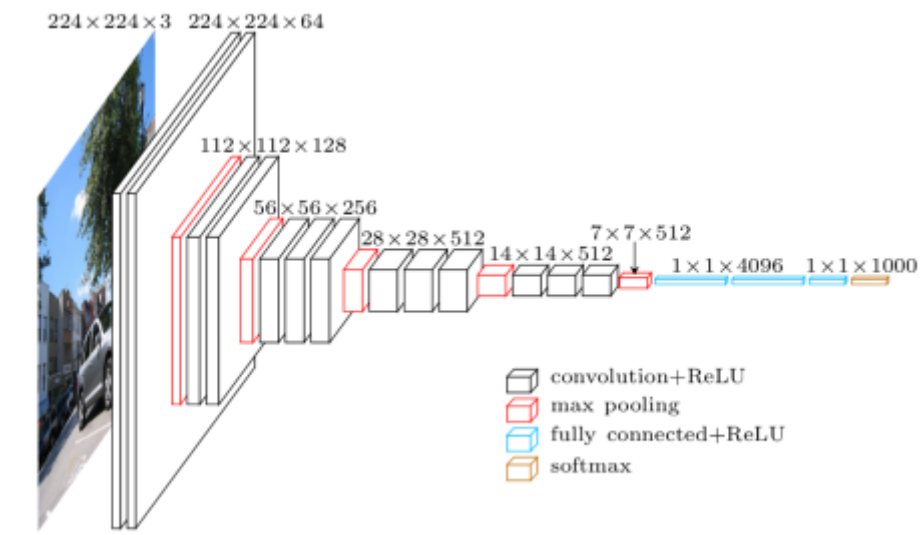

Figure 2.5: VGG neural network architecture

Vision Transformers (ViT)

Vision Transformers (ViTs) are an innovative approach to image processing that uses self-attention mechanisms inspired by the Transformer architecture, which was originally created for natural language processing applications.This architecture marks a significant shift from traditional Convolutional Neural Networks (CNNs) in terms of image processing for recognition tasks. At its core, ViTs break down input images into fixed-size patches, which are subsequently flattened and processed by a series of transformer encoder layers. These transformer layers allow the model to capture global dependencies across image patches, resulting in a more comprehensive comprehension of visual content. ViT divides the input image into fixed-size, non-overlapping patches, such as dividing a 224x224 image into 16x16 blocks. Each patch is then flattened into a 1D vector and linearly projected into a higher-dimensional embedding space, yielding a series of patch embeddings that serve as input tokens for the model. We augment these patch embeddings with positional encodings to integrate spatial information. These positional encodings are gained during training and assist the model in understanding the relative placements of patches, making up for the Transformer architecture's lack of innate spatial awareness. The ViT design employs many Transformer encoder layers, each with a multi-head self-attention mechanism and a feedforward neural network. The self-attention mechanism enables the model to represent links between patches by computing a weighted total of

all patch embeddings, highlighting important patches and taking into account both local and global contexts. Each encoder layer's output is normalized and augmented using residual connections to stabilize training and improve gradient flow.

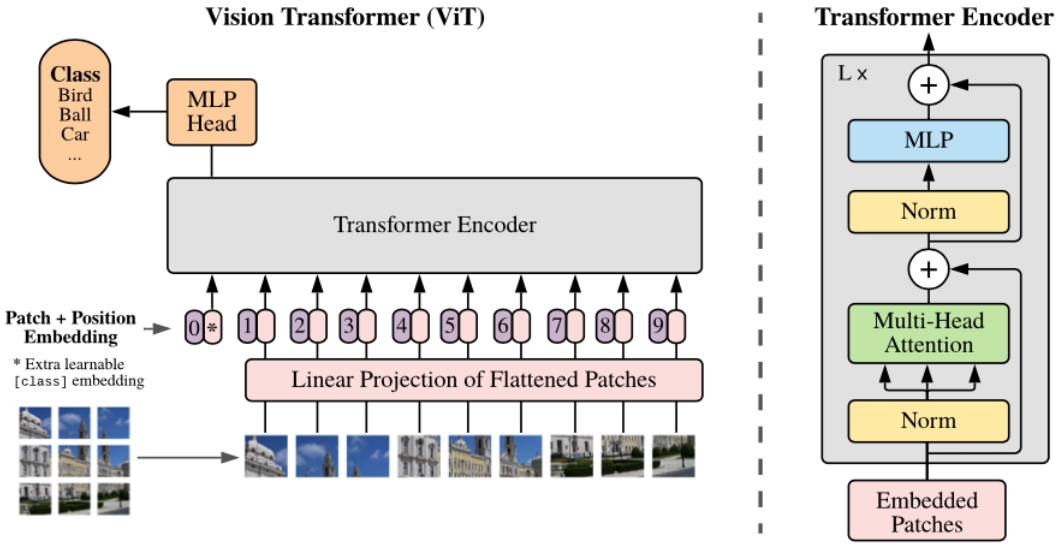

Figure 2.6: Vision transformers architecture

The Vision Transformer excels in capturing global context through self-attention, distinguishing itself from CNNs, which use convolutional and pooling layers for hierarchical feature extraction. Despite the high computing needs of the self-attention mechanism, ViT outperforms other image recognition benchmarks when pre-trained on big datasets and fine-tuned for specific tasks. This design demonstrates that a pure transformer model, without convolutional layers, can deliver cutting-edge results in computer vision, representing a watershed moment in the field.

2.4 Model Training

Sequential model

We utilized the Keras package to construct a Convolutional Neural Network (CNN) architecture. The model is developed with the Sequential API, which allows for a linear stack of layers. The model begins with a convolutional layer of 32 3x3 filters, which is then followed by the ReLU activation function. The convolutional layer is followed by a max-pooling layer with a pool size of 2x2, which reduces the spatial dimensions of feature maps and introduces translation in-

variance. The model is performed twice more, with successive convolutional layers containing 64 and 128 filters, respectively. The result is flattened with the Flatten layer, which turns multidimensional feature maps into one-dimensional vectors.The output is then processed through two Dense layers, with the last Dense layer representing the number of classes in the classification issue.

The model is compiled using the Adam optimizer, the categorical cross-entropy loss function, and the accuracy metric. During training, the model is fitted to the training data via Keras' fit() method. The training procedure lasts 10 epochs, and performance is assessed on test data generated separately after each epoch. Training and assessment measures, such as loss and accuracy, are recorded for each epoch, allowing for performance monitoring of the model. Total params:

Table 2.1: Sequential model summary

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 46, 46, 32)	896
ax_pooling2d (MaxPooling2D)	(None, 23, 23, 32)	0
conv2d_1 (Conv2D)	(None, 21, 21, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 10, 10, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73856
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
dense_1 (Dense)	(None, 27)	3483

359,003 (1.37 MB)
Trainable params: 359,003 (1.37 MB)
Non-trainable params: 0 (0.00 Byte)

VGG-19

The VGG19 is a deep convolutional neural network architecture designed for image classification applications. It preprocesses input photos, converts them to RGB format, and scales them to 224x224 pixels. The preprocessed photos and labels are saved as NumPy arrays for easy loading during training and assessment. The model is initial-

ized with pre-trained weights from the ImageNet dataset via transfer learning, allowing it to use previously learned characteristics from a large dataset. The model is built using the Adam optimizer, a sparse categorical cross-entropy loss function, and the accuracy metric. A model checkpoint is created to save the best-performing model during training, and the model is trained using the model.fit() function, which iterates over the training data in batches of 32 samples for 15 epochs. The model's performance is tested against validation data, and if validation accuracy increases, the model weights are recorded. Regular adjustments to the model's parameters are made based on the obtained gradients and optimization procedure. The model's performance on test data is assessed using the model.evaluate() function, and the best-performing model is translated to TensorFlow Lite for deployment on mobile or embedded devices.

Vision Transformers

To ensure reproducible results, the Vision Transformer (ViT) model is trained using random seeds. Torchvision loads a pre-trained ViT-Base model, which was previously trained on the ImageNet dataset. To make use of the model's feature extraction capabilities, the base parameters are locked. A new linear layer was created for our target dataset, which has 27 classes. We loaded thetTraining and testing data are from specific folders, preprocessed with the necessary transformations, then packed as PyTorch DataLoader objects.

The model was trained utilizing an Adam optimizer, a 1e-3 learning rate, and a cross-entropy loss function. The training loop iterated over a set number of epochs, computing the loss, backpropagating, and recording training metrics. The model's performance on test data was assessed, whereby we computed metrics such as loss and accuracy. Following completion, the fine-tuned ViT model's state dictionary was stored to a file, and assessment metrics such as accuracy, precision, recall, and F1 score were calculated using the test dataset. The model's classification skills were displayed by doing predictions on bespoke pictures.

EfficientNet

EfficientNet is a class of convolutional neural networks that effectively scale up models by balancing network depth, breadth, and resolution via a compound scaling algorithm. This design offers cutting-edge speed while remaining computationally effective, making it appropri-

ate for a wide range of applications, including mobile devices and large-scale cloud settings. EfficientNet begins with the EfficientNet-B0 model and gradually increases the depth (number of layers), breadth (number of channels), and resolution (input picture size). The introduction of a compound coefficient enables simultaneous scaling along these three dimensions, guaranteeing that model complexity increases at an acceptable rate while avoiding overfitting or underfitting. EfficientNet's key features are compound scaling, mobile inverted bottleneck convolution (MBConv), and swish activation. EfficientNet offers more accuracy and efficiency than earlier CNN topologies, making it a good candidate for a variety of image identification workloads.To use EfficientNet, we first import the required libraries and then load the pre-trained EfficientNet model from the TensorFlow Hub. We next prepare the input data by scaling and normalizing the photos. Next, we build a sequential model with the EfficientNet model as its foundation, followed by additional layers for classification (e.g., GlobalAveragePooling2D, Dense). We assemble the model with the proper loss function and optimizer, then train it on the provided dataset. During training, we use callbacks and validation data to assess the model's performance.

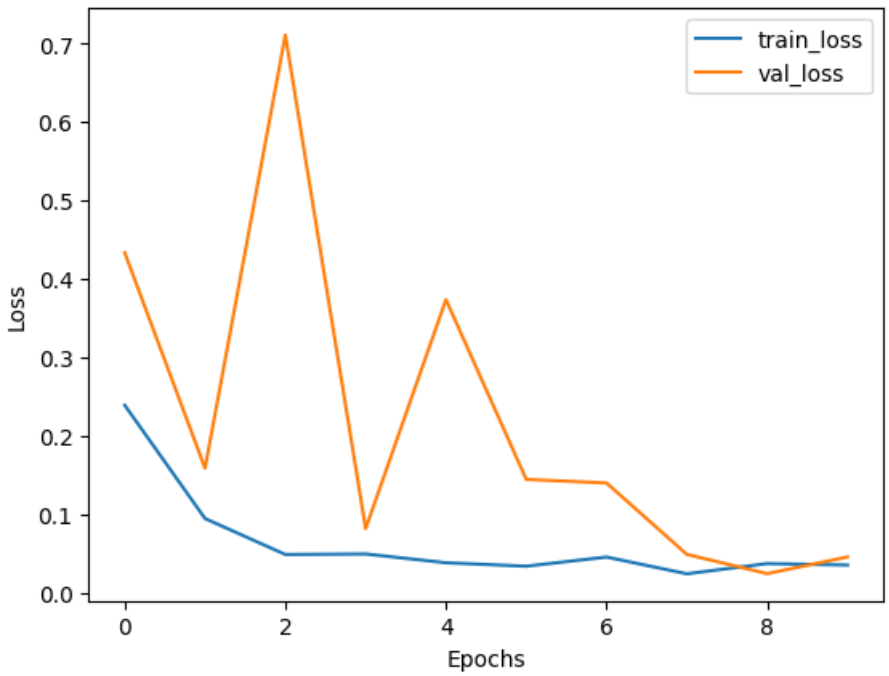

Figure 2.7: Efficientnet train_loss and evaluation_loss curve

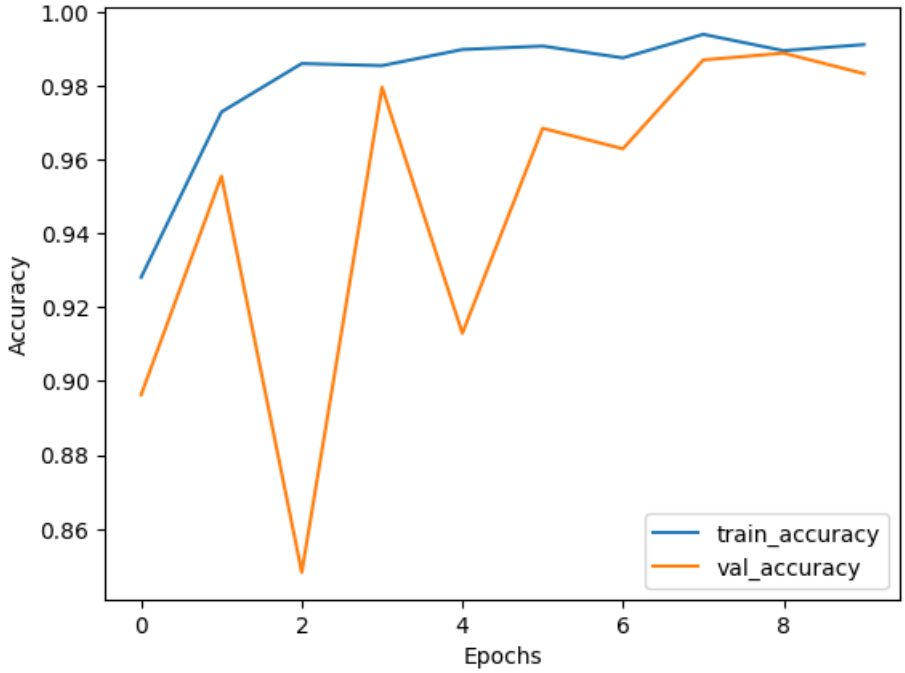

Figure 2.8: Efficientnet train_accuracy and evaluation_accuracy curve

ResNet

ResNet (Residual Network) is a popular deep learning architecture recognized for introducing residual learning, which solves the vanishing gradient problem and allows for the training of extremely deep networks. ResNet's main innovation is the usage of residual blocks, in which identity mappings (shortcut connections) bypass one or more levels. ResNet design comprises of Residual Blocks with shortcut connections and, in more advanced variants, Bottleneck Blocks for parameter efficiency. ResNet variations include ResNet-18 and ResNet-34 (basic residual blocks), as well as ResNet-50, ResNet-101, and ResNet-152 (bottleneck residual blocks). ResNet's capacity to train very deep networks has resulted in widespread use and success in a variety of computer vision applications, including picture classification, object identification, and segmentation.

Video Based Dataset

We used ResNet to recognize words in American Sign Language, with the goal of bridging the communication gap between the deaf and hearing communities. The researchers applied the ResNet architecture, which is recognized for its efficiency and efficacy in image classification tasks, to a subset of the WLASL video collection, the biggest dataset of its type. The objective was to create a model capable

of properly reading ASL signs from the chosen subset, allowing for the development of new communication tools that could improve interactions between people from various linguistic backgrounds. We used a systematic method, loading necessary Python packages such as TensorFlow Keras and subjecting the video data to stringent preparation approaches. They then fine-tuned a pre-trained ResNet model for word-level ASL recognition from the WLASL subset, taking advantage of its robust design and customizing it to their specific goals within computational restrictions. The training approach entailed running sessions on a specific portion of the WLASL dataset, closely evaluating the model's performance, and making tweaks and improvements to achieve optimal results. Initial studies provided excellent results, proving ResNet's capability in processing video data for ASL recognition, despite a restricted dataset. This study establishes the framework for future research and development in sign language comprehension systems, with the objective of enhancing communication solutions that enable persons to overcome linguistic hurdles.

Table 2.2: VGG-19 model summary

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 250880)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
Total params:		143667240 (548.05 MB)
Trainable params:		143667240 (548.05 MB)
Non-trainable params:		0 (0.00 Byte)

Table 2.3: ViT model summary

Layer (type)	Output Shape	Param #
conv_proj	(None, 224, 224, 3)	36,864
encoder layers (12)	Multiple	Multiple
heads	(None, 27)	20,763
Total params:	57,627	
Trainable params:	57,627	
Non-trainable params:	0	

Table 2.4: Video results

Class	Precision	Recall	F1-score	Support
Bye	0.0	0.0	0.0	1
Car	0.5	1	0.67	1

Chapter 3

Major Research Findings

3.1 Model Evaluation

We used a number of evaluation metrics in order to assess the performance of our models which included accuracy, loss, precision, recall, F1-score, and as illustrated in the table. The evaluations enabled us to understand how accurately the model is predicting and classifying validation data. The confusion matrix was used in identifying possible misclassifications showing visual representation of classification results.

Table 3.1: Performance of the models

Metrics	Results		
	VGG-19	ViT	Mobilenet
Precision	0.9782409	0.9294	0.99
Recall	0.977778	0.9285	0.99
F1 Score	0.977798	0.9285	0.99
Accuracy	0.977778	0.9285	0.99

3.2 Explainable AI Results

Explainable AI refers to an AI model's projected impact and potential biases. It contributes to model correctness, fairness, transparency, and results in AI-powered decision making. Explainable AI is critical for

enterprises to create trust and confidence in AI models before bringing them into production and taking a responsible approach to AI development. As AI progresses, humans are challenged to grasp and retrace the algorithm's steps, resulting in "black box" models that are hard to explain. Engineers and data scientists who develop these models are unable to comprehend or explain the algorithms' activities. Explainability has various benefits, including guaranteeing system operation, satisfying regulatory criteria, and allowing individuals impacted by a decision to contest or amend the conclusion.

We used the SHAP, GradCAM, LIME, and Integrated Gradients techniques to gain insight into ViT and VGG-19 decision-making processes. LIME provided local explanations by identifying the most significant aspects of an image that influenced the model's predictions. By visualizing these explanations, we gained a deeper understanding of the features that each model considers while categorizing. Integrated Gradients assigned relevance values to individual pixels, allowing us to observe how different image sections influenced the final forecast. These interpretability methodologies helped us understand the inner workings of both models, providing valuable insights into their decision-making processes.

3.2.1 LIME

Specifically, LIME stands for Local Interpretable Model-agnostic Explanations. LIME focuses on developing local surrogate models that can explain individual forecasts. Local surrogate models are interpretable models used to explain specific predictions from black box machine learning methods. Surrogate models are designed to mimic the predictions of the underlying black box model. Rather of training a global surrogate model, LIME trains local surrogate models. LIME is model-agnostic, meaning it may be used with any machine learning model. The approach tries to understand the model by varying the input of data samples and observing how predictions change. Model-specific techniques seek to perceive a black model machine learning model by investigating its fundamental components and how they interact. LIME supports local interpretation of models. LIME modifies a single data sample by modifying the feature values, and then investigates the effect on the outcome. The most often asked question is probably why this prognosis was made or what conditions led to it. LIME aims to explain any classifier's predictions in an understandable

manner, with a focus on local fidelity. The technique includes producing data variants, creating a new dataset by perturbing the original instance of interest, training an interpretable model, and weighting the samples based on their similarity to the original instance. This ensures that the local approximation is more precise and true in reference to the specific instance being addressed.

The mathematical formula for the explanation model is created by minimizing a loss function, which measures how well the explanation fits the original model's forecast. The objective is to develop a model that minimizes this loss while maintaining model simplicity. This may be expressed quantitatively, as shown in the expression:

$$\text{explanation}(x) = \underset{g \in \mathcal{G}}{\arg} \, \mathcal{L}(f, g, \pi(x)) + \Omega(g)$$

The explanation model for instance x is the model g (e.g., linear regression) that minimises the loss function L (e.g., mean squared error). It evaluates how well the explanation matches the prediction of the original model f (e.g., VGG 19), while keeping the model complexity Ω(g) low (e.g., fewer features). G denotes the family of credible explanations. LIME optimizes the loss component of this equation by selecting the instance of interest, perturbing the dataset, generating predictions, weighting the samples, training the interpretable model, and then interpreting the local model.

LIME's approach to understanding deep learning model predictions is to take a single instance, disrupt it by producing permuted instances, and then apply the deep learning model to these samples.If LIME then trains an interpretable surrogate model, comparable to a sparse linear model, to simulate the deep model's outputs locally surrounding the instance under study.

The trained surrogate model generates an interpretable representation of how different super pixels influence the deep learning model's prediction for that instance, which is often shown as a heat map. The lime image module from the LIME package evaluates the output of deep convolutional neural networks trained on image data. Applying LIME to various scenarios enables the investigation of the explanations provided by local surrogate models for the deep neural network's decisions, auditing the model's operation, identifying any biases or artifacts being learned, and developing more transparent and trustworthy

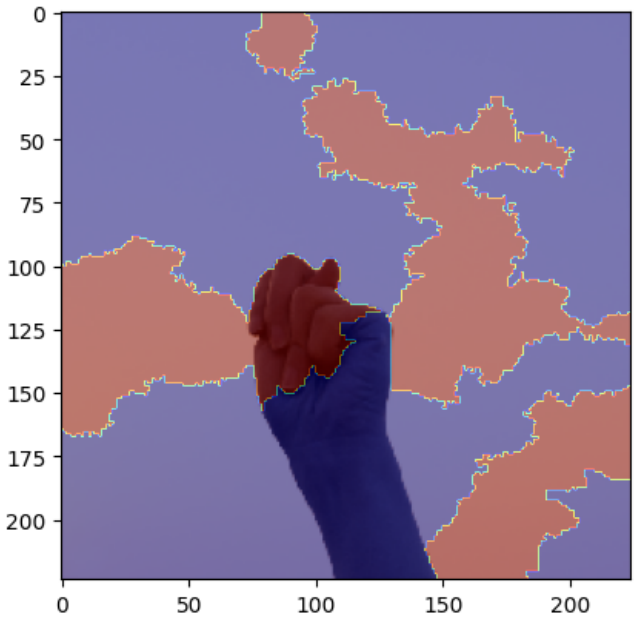

Figure 3.1: Explainability using LIME

artificial intelligence systems. LIME's interpretability complements deep learning models' excellent prediction accuracy.

3.2.2 SHAP

SHAP (Shapley Additive ExPlanations) [14] is a machine learning method that provides a significance value (SHAP value) to each feature in a model based on Shapley values. These numbers describe a feature's average contribution to the forecast, ensuring that contributions are distributed fairly across all features. SHAP's main notion is based on Shapley values, a cooperative game theory concept in which the "game" is the model's prediction objective and the "players" are the characteristics of the input data.

SHAP is model-agnostic, which means it can work with any machine learning model, such as deep learning, decision trees, and ensemble models. It does not require access to the model's internal workings, hence it is suited for a wide range of applications. SHAP incorporates numerous processes, including instance selection, dataset perturbation, model prediction, sample weighting, interpretable model training, and explanation extraction. The process of determining the optimal local surrogate model may be stated numerically as shown in the expression below:

$$\text{explanation}(x) = \underset{g \in \mathcal{G}}{\text{argmin}}\ \mathcal{L}(f, g, \pi(x)) + \Omega(g)$$

Where f is the original model, g is the interpretable surrogate model, L is the loss function, πx is the proximity measure, and Ω(g) represents the surrogate model's complexity, promoting simpler explanations. We used SHAP (Shapley Additive Explanations) to explain the predictions of a VGG19 model, which is a deep convolutional neural network trained on the ImageNet dataset. The SHAP employs a game-theoretic method to give insights into how specific attributes affect model predictions. To begin, the relevant libraries were loaded, including Keras, which is used to load the VGG19 model and preprocess the inputs. The VGG19 model was loaded with pre-trained ImageNet weights, which included fully linked layers for classification.

A section of the dataset was used, with two photographs chosen for explanations. A JSON file providing class names linked with the labels was also loaded to aid interpretation. A function was built to translate input pictures to outputs from a given layer inside the network, allowing SHAP to interact with the VGG19 model. A SHAP 'GradientExplainer' object was created, with the input and output layers of the VGG19 model and the input data mapped to the appropriate layer. The 'GradientExplainer' computes the SHAP values, which indicate the significance of each feature within the context of the model's predictions.

The derived SHAP values provide a thorough explanation of how each characteristic influenced the model's predictions. The explanations were made more understandable by translating class indices to their respective names based on the loaded class names. Visualizing the SHAP explanations provided useful insights into the model's decision-making process by highlighting the most important elements that drove its predictions. By combining SHAP with the VGG19 model, the researchers obtained a better grasp of how the model analyzes and classifies pictures, increasing transparency and trust in the program's predictions.

Figure 3.2 shows SHAP results, explaining our custom model predictions. The model predicted a label for each image input as shown above, and SHAP explained why the model provided a given output by using a mask to highlight the most relevant features with red and the least important features with blue. This provided more insight

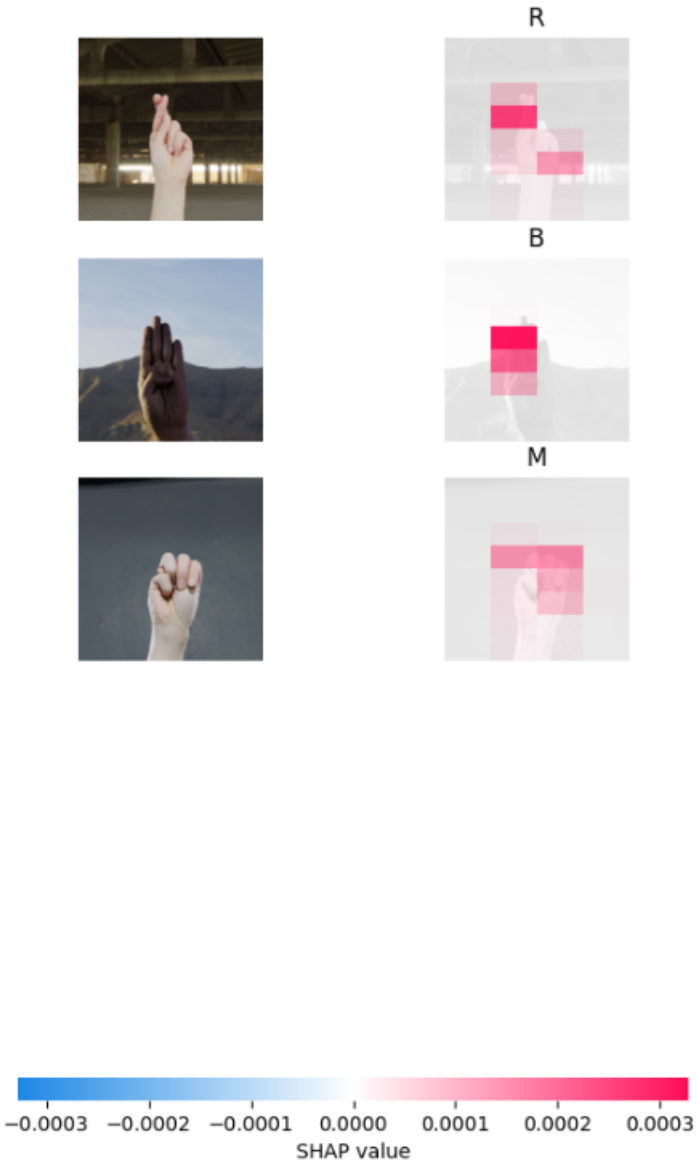

Figure 3.2: Explainability using SHAP

into the models' decision making process, enhancing transparency and interpretability.

3.2.3 GradCAM

GradCAM is a technique for analyzing essential sections of an image so that a convolutional neural network (CNN) can make decisions. It works by computing the gradient of the class score in relation to the final convolutional layer's feature maps, which are then weighted to identify the sections of the image that have a significant impact on the prediction. GradCAM has numerous steps, including a forward pass, specifying the target class, a backward pass, global average pooling, weight feature mapping, ReLU activation, and normaliza-

tion.GradCAM employs the gradient flowing in the final convolutional layer to assign priority values to each neuron, as shown in the expression below, for every model's decision.

$$\alpha_k^c = \underbrace{\frac{1}{Z}\sum_i\sum_j}_{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

We apply ReLU to the linear combination of maps because we are only interested in features that have positive effect on the class of interest, i.e. pixels whose intensity should be increased to increase the yĉ. We then normalize all the values in the grid with a min-max normalization, and plot the tensor with a heatmap. Each of these methods improves on the original GradCAM process by improving resolution, removing gradients, or employing PCA, resulting in more accurate and clear visual explanations of CNN predictions.

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\underbrace{\sum_k \alpha_k^c A^k}_{\text{linear combination}}\right)$$

HiResCAM seeks to generate higher resolution heatmaps by reshaping the gradients to match the geometry of the feature map, resulting in a more accurate representation and element-wise multiplication of the feature maps using the reshaped gradients. ScoreCAM improves on GradCAM by eliminating gradients and producing heatmaps based on the activation maps of the final convolutional layer. EigenGradCAM generates heatmaps by combining Principal Component Analysis (PCA) with GradCAM. It consists of a forward pass to identify the target class, a backward pass, PCA application, component weighting, and normalization.

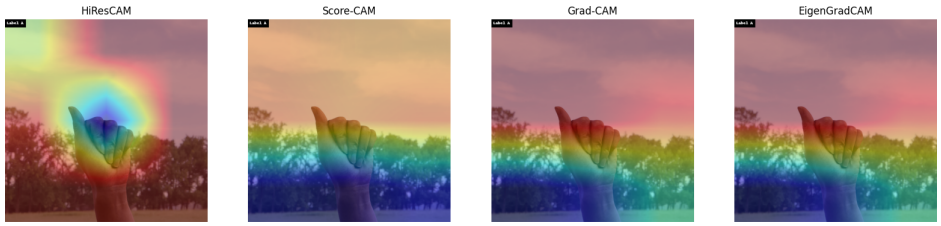

Figure 3.3: Perfomance of various CAMs

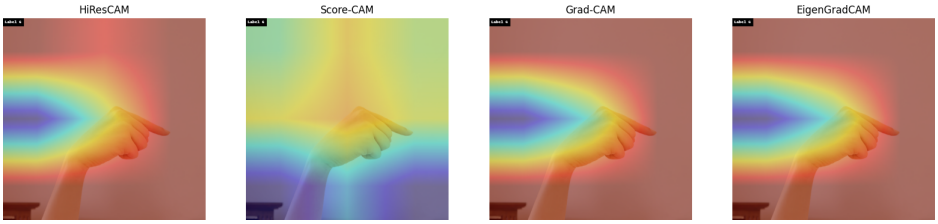

Figure 3.4: Performance of various CAMs

Figure 3.5: Gradcam on mobilenet

3.2.4 Integrated gradients

Integrated Gradients is a method for attributing a neural network's prediction to its input features. It entails computing the integral of the output gradients with respect to the input along a path from a baseline to the input. This method aids in understanding the contribution of each input characteristic to the model's prediction, making it an effective tool for comprehending Vision Transformers.

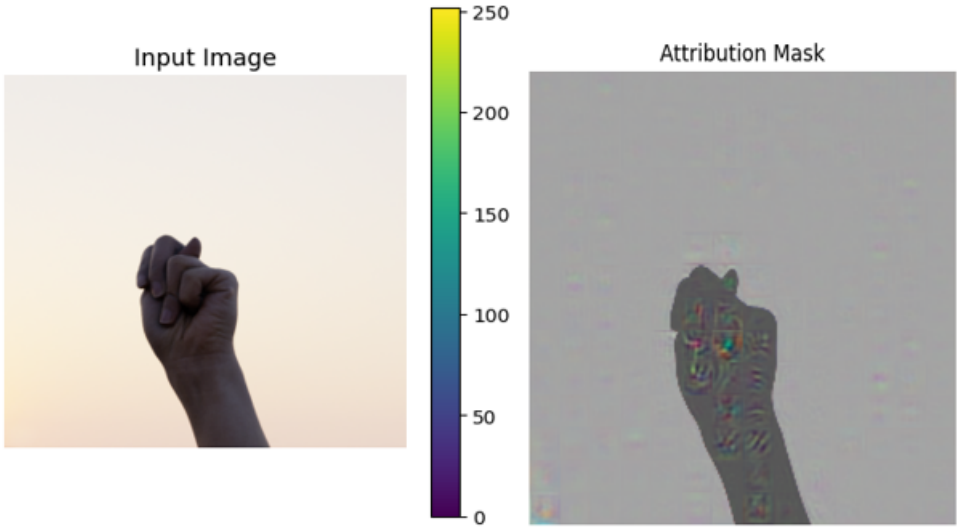

Figure 3.6: ViT explainability using Integrated gradients

Integrated Gradients provide an attribution map that emphasizes the importance of each pixel in the input picture in the model's prediction. For Vision Transformers, this can show how different components of the picture impact the final decision, offering a fine-grained explanation for the model's behavior. This method uses the gradient integral to create a more plausible explanation. It requires no changes to the original model, making it widely adaptable. However, the baseline must be carefully set since it might impact the outcomes. It is also computationally costly to create gradients for several interpolated inputs.

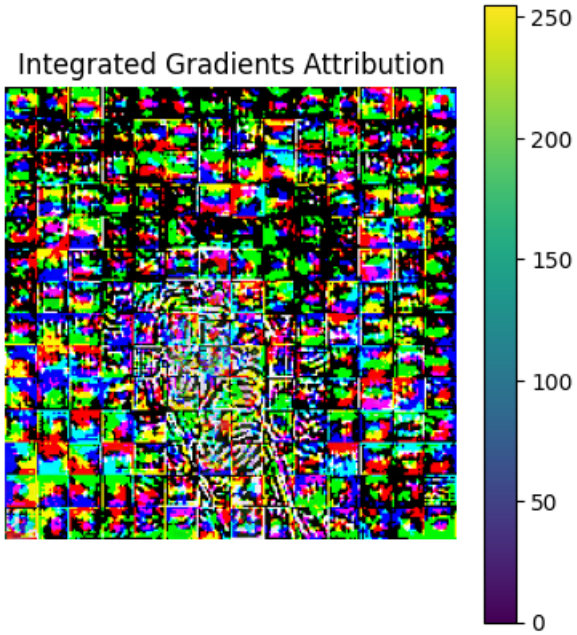

Figure 3.7: Integrated gradients attribution

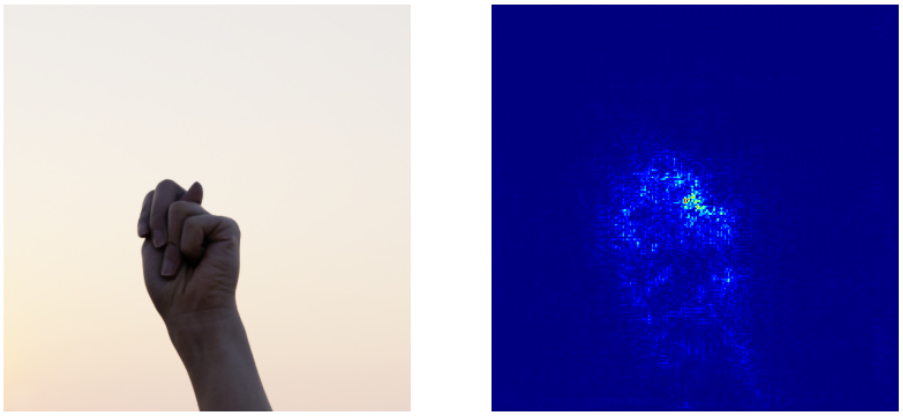

Figure 3.8: saliency map of Mobilenet

3.3 Practical Implications

The use of SHAP (SHapley Additive Explanations) increases the transparency of a sign language recognition model. This enables consumers to comprehend the reasoning behind the model's conclusions, which boosts trust and confidence in the system. This openness makes the concept more accessible and valuable to a larger audience. Most research on sign language models has concentrated on constructing models rather than incorporating them into actual applications. However, the absence of integration renders the models almost worthless in real-world circumstances. The study fills this gap by integrating a sign language model into a mobile application, resulting in intuitive and user-friendly interactions. This real user interface assures that the model is more than simply a theoretical concept, but also a practical tool for everyday circumstances, making sign language recognition more accessible and valuable to a wider audience.

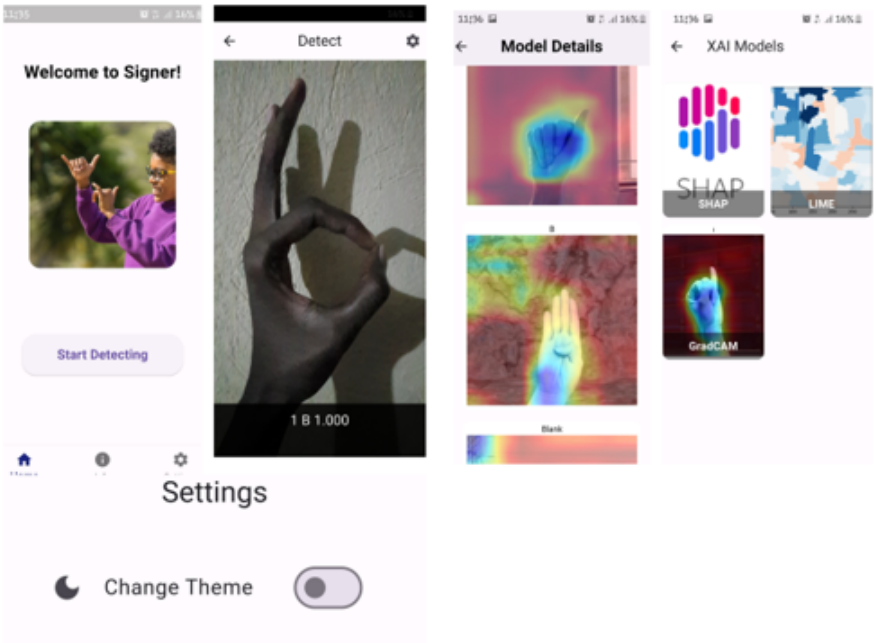

Figure 3.9: Some screens in the mobile application

3.3.1 Research Limitations

Our research faced several limitations, primarily due to the constraints of the available datasets and computational resources. Firstly, we were unable to run the model on the entire video-based dataset, which would have provided more comprehensive insights into dynamic sign

language recognition. This limitation arose from the insufficient computational resources required to process and analyze video data effectively. Secondly, the unavailability of a Ugandan Sign Language dataset posed a significant challenge. The process of creating a new dataset involves rigorous regulatory and procedural requirements, which are both time-consuming and costly. The absence of an existing dataset for Ugandan Sign Language hindered our ability to train and validate the model on locally relevant data, limiting the scope and applicability of our research to the Ugandan context.

3.3.2 Originality/Value

We have developed an AI model that can recognize and translate sign language into text, providing explanations for their decisions. This model uses techniques like SHAP, LIME, GRADCAM and Integrated Gradients to enhance transparency and explainability, providing users with insights into how the models operate. This helps provide interpretable explanations for the models' predictions, allowing users to evaluate the system's performance, identify potential biases, and build trust in the technology. This approach encourages the adoption of AI in a responsible and accountable manner, ensuring the technology is effective, trustworthy, and fair. The model's performance is compared with existing methods, providing a benchmark for future research and facilitating advancement in the field. The model is also converted into TensorFlow Lite (TFLite) and integrated into a mobile application, ensuring real-time and user-friendly interaction. This practical implementation bridges the gap between theoretical research and real-world application.

Chapter 4

Conclusion and Future Research Work

4.1 Challenges in Current Models

We aimed to improve the accuracy and explainability of a real-time sign language-to-text model. A high-quality dataset was utilized to represent numerous signals and their complexity in sign language. They carefully preprocessed and enhanced the data to improve its quality and diversity. They tested many models, including VGG19 and Vision Transformers, to identify which performed best for object-detecting tasks. VGG19 outperformed expectations, with greater accuracy, precision, and recall rates. It also exhibited a decreased misclassification rate, showing that it could deal with complicated visual patterns more efficiently. The study also emphasized model explainability to improve transparency and user confidence. Techniques such as SHAP, LIME, and Grad-CAM were employed to get insight into the model's decision-making process. The model was subsequently translated to TensorFlow Lite (TFLite) and incorporated into a smartphone application, allowing for real-time, intuitive, and user-friendly interactions. This integration closes the gap between theoretical study and practical application, transforming the sign language recognition model into a viable tool for real-world use.

4.2 Future work

We aim to broaden the dataset by including a variety of sign languages, geographical variances, and complicated motions to improve the model's generalizability and resilience. This might be accomplished by collaborating with language specialists and communities, as

well as employing data augmentation techniques to replicate a variety of settings and environments. Real-time performance optimization is critical for practical applications, since it reduces the computing burden without sacrificing precision. Model pruning, quantization, and lightweight designs such as MobileNets or EfficientNets can be used to make the model more accessible to a wider audience. Multimodal integration can increase sign language identification accuracy and reliability by combining hand form, movement trajectory, facial expressions, and body position. Advanced sensing technologies and deep learning algorithms should be researched in this regard. The work also intends to create a Ugandan sign language dataset utilizing the Lexset synthetic ASL alphabet and the MS-ASL dataset. This dataset will be gathered and utilized to train a model capable of translating Ugandan sign language into text, therefore addressing a key research need and contributing to the preservation and accessibility of Ugandan sign language.

Acknowledgments

We would like to extend our sincere gratitude for full facilitation of our effort to the RISE Seed Fund at Makerere University College of Computing and Information Sciences. We also thank Mr. Ggaliwango Marvin, our supervisor, for his consistent guidance.

Group CS24-22

June, 2024
Makerere University

References

[1] W. Aly, S. Aly, and S. Almotairi, "User-Independent American Sign Language Alphabet Recognition Based on Depth Image and PCANet Features," IEEE Access, vol. 7, pp. 123138–123150, 2019, doi: https://doi.org/10.1109/access.2019.2938829.

[2] "HON. ASAMO HELLEN GRACE(MP) MINISTER OF STATE FOR DISABILITY AFFAIRS." Accessed: Nov. 26, 2023. [Online]. Available: https://parliamentwatch.ug/wp-content/uploads/2022/09/Statement-on-the-International-Deaf-Awareness-Week-19th-to-23rd-S-Asamo-Hellen-Grace-MP-Minister-of-State-for-Disability-Affairs.pdf

[3] J. Bora, Saine Dehingia, Abhijit Boruah, Anuraag Anuj Chetia, and Dikhit Gogoi, "Real-time Assamese Sign Language Recognition using MediaPipe and Deep Learning," vol. 218, pp. 1384–1393, Jan. 2023, doi: https://doi.org/10.1016/j.procs.2023.01.117.

[4] R. K. Pathan, M. Biswas, S. Yasmin, M. U. Khandaker, M. Salman, and A. A. F. Youssef, "Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network," Scientific Reports, vol. 13, no. 1, p. 16975, Oct. 2023, doi: https://doi.org/10.1038/s41598-023-43852-x.

[5] A. Barredo Arrieta et al., "Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," Information Fusion, vol. 58, no. 1, pp. 82–115, Jun. 2020, doi: https://doi.org/10.1016/j.inffus.2019.12.012.

[6] 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, doi: https://doi.org/10.1109/access.2021.3110912

[7] "Interpretability of machine learning: what are the challenges in the area of automated decision-making processes?," Wavestone. https://www.wavestone.com/en/insight/interpretability-machine-learning/

[8] IBM, "What is explainable AI? — IBM," www.ibm.com. https://www.ibm.com/topics/explainable-ai#:~:text=Explainable%20artificial%20intelligence%20(XAI)%20is

[9] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Advances in Neural Information Processing Systems 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774.

[10] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in 2017 IEEE International Conference on Computer Vision (ICCV), Oct. 2017, pp. 618–626. doi: 10.1109/ICCV.2017.74.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, Aug. 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778.

[12] "Synthentic ASL Alphabet," www.kaggle.com. https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet

[13] "MS-ASL," Microsoft Research, Aug. 05, 2019. https://www.microsoft.com/en-us/research/project/ms-asl/

[14] mikesuperman, "SHapley Additive exPlanations ou SHAP : What is it ?," Data Science Courses — DataScientest, Mar. 09, 2023. https://datascientest.com/en/shap-what-is-it

[15] "Normalization (image processing)," Wikipedia, Apr. 16, 2020. https://en.wikipedia.org/wiki/Normalization_(image_processing)

[16] “VGG16 - Convolutional Network for Classification and Detection,” Neurohive.io, Nov. 21, 2018. https://neurohive.io/en/popular-networks/vgg16/

[17] J. N. Jan 26 and Read 2020 6 M., “Why should I do pre-processing and augmentation on my computer vision datasets?,” Roboflow Blog, Jan. 26, 2020. https://blog.roboflow.com/why-preprocess-augment/