

Sign Language Detection Using CNN and LSTM Based Model

Bhavya Chauhan¹, Akshunya Banjarey², Vibhor Sharma³

Department of Information Technology

Maharaja Agrasen Institute of Technology, Delhi, India

Abstract: In the context of our current work, we have taken a major step in examining and contrasting the effectiveness of several machine learning models for sign language recognition. Convolutional Neural Network (CNN) model building was the focus of our minor project, which resulted in the successful publication of a research paper [6]. Our method was validated by this model, which showed significant accuracy in recognizing sign language.

We wanted to build a Long Short-Term Memory (LSTM) model for the same reason as we moved on to our larger project. The aim was not only to construct an additional efficient model but also to conduct a comparative analysis between the LSTM and the CNN model that was previously constructed. This comparison included a thorough assessment of both models based on a number of variables, including accuracy, loss, speed, and others.

We can now confidently state that we have created an LSTM model for sign language recognition that is fully operational. In addition, we have compared our CNN model with our LSTM model, assessing both models according to training accuracy, training loss, validation accuracy, validation loss, and training duration. This development opens the door for more research in this area and represents an important turning point in our study. Our research will be strengthened by the solid foundation our findings provide.

Keywords: Convolutional Neural Network, Long Short-Term Memory, Sign Language Detection, Training, Validation

1. Introduction:

For the deaf and hard of hearing community, sign language is an essential communication tool that promotes their engagement and social integration. Nevertheless, even with its importance, sign language accessibility might be restricted, especially in situations when impediments to communication are present. By allowing automated gesture recognition and interpretation, the incorporation of machine learning techniques into sign language detection systems has demonstrated potential in closing these gaps in recent years. In order to detect sign language, this research study examines and contrasts several machine learning techniques. It places particular attention on the employment of convolutional neural networks (CNNs) and long short-term memory (LSTM) models in the context of Sign Language. By exploring the effectiveness of these models, we aim to provide insights into their performance, limitations, and potential applications in real-world scenarios. This research is motivated by the need to advance sign language detection technology, ultimately enhancing accessibility and inclusivity for the deaf and hard of hearing community.

2. Literature Review:

The field of sign language detection has seen significant progress through the application of machine learning models, especially convolutional neural networks (CNN) and long short-term memory networks (LSTM). A comprehensive review of the literature reveals a variety of approaches and methods used in this field.

A research paper titled “Real-time sign language detection using CNN” [1] proposed a deep learning-based approach for sign language detection, aiming to narrow the gap communication between normal people and deaf people. The researchers used a custom CNN model trained on a dataset containing 11 signal words, achieving 98.6% accuracy.

Another study, “Sign Language Detection Using LSTM,” [2] focuses on using LSTM to detect sign language. The article highlights the difficulties that ordinary people face in understanding sign language and proposes LSTM as a solution to these challenges.

The article “Sign language recognition using deep learning models of long-term and short-term memory” [3] presents a model using MediaPipe Holistic and LSTM to recognize language signs for people with disabilities. The model achieves 98.50% accuracy in detecting Standard American Sign Language.

In the article “CNN and Stacked LSTM Model for Indian Sign Language Recognition” [4], the authors proposed a deep learning model for sign language recognition using CNN and LSTM. The architecture used CNN as a pre-trained model for feature extraction and fed it into LSTM to collect spatio-temporal information.

The paper “Machine learning-based sign language recognition: a review of and its research areas” [5] provided a comprehensive review of about 240 different approaches exploring recognition sign language to recognize multilingual signs.

These studies demonstrate the potential of CNN and LSTM models in sign language detection and recognition. They also emphasized the importance of comparative studies between these models to determine the most effective approach to this task. The results of these studies will significantly contribute to the development and improvement of sign language detection models.

3. Methodology:

Working of CNN Model

Convolutional Neural Networks (commonly referred to as CNNs or ConvNets) [6] are a subclass of neural networks that are particularly well-suited for processing data with grid-like topologies, such as image data. A binary representation of visual data is a digital picture. It has a grid-like arrangement of pixels, and the brightness and colour of each pixel are represented by pixel values. The moment the human brain sees a picture, it begins processing a vast quantity of data. The complete visual field is covered by the connections between each neuron, each of which operates in its own receptive field.

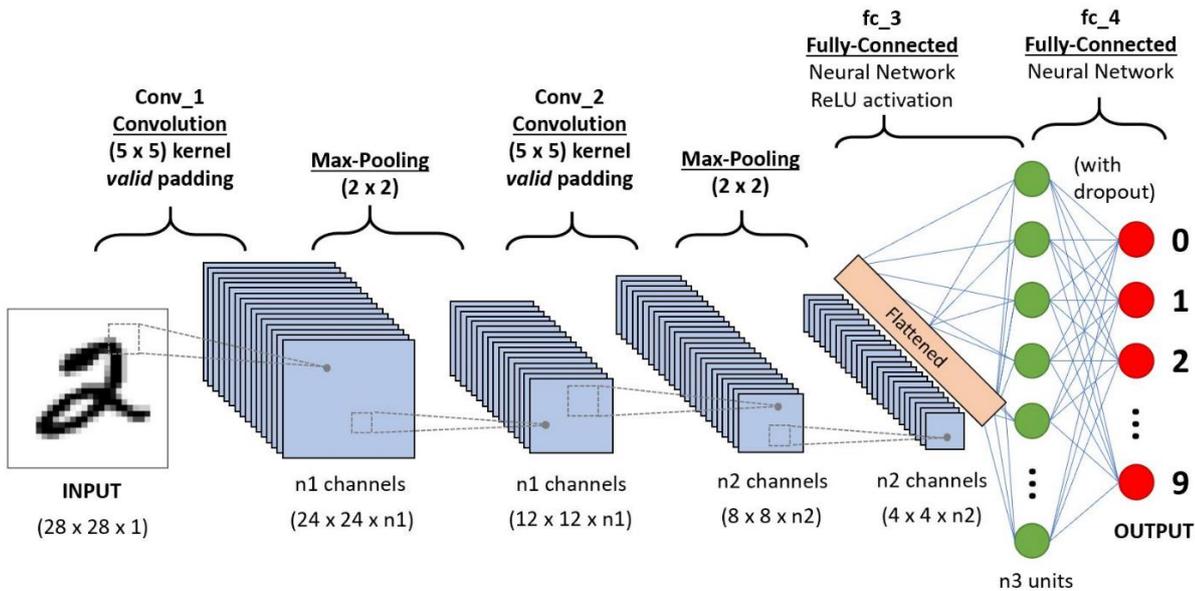


figure 1: Convolutional Neural Networks (CNNs) [7]

Collection of data : We started collecting data by communicating with a webcam to capture images. The python file creates a folder structure to organize the collected data, assigning each alphabetically from A to Z to the corresponding subfolder. By pressing the key corresponding to a specific letter, the current image will be saved as a grayscale image resized to 48 x 48 pixels. This systematic data collection process ensures the availability of a diverse and well-organized data set, which is important for model training.

Feature Extracion : The system described orchestrates real-time American Sign Language (ASL) gesture detection. It begins by importing essential libraries, including those for loading a pre-trained Convolutional Neural Network (CNN) model and for real-time video processing using OpenCV. The pre-trained CNN model is loaded from JSON and HDF5 files, equipped to detect ASL gestures. A crucial aspect of the system is the feature extraction process, where video frames are converted to numpy arrays, reshaped to fit the model's input size, and pixel values normalized. In real-time, frames are captured from the webcam, preprocessed to match the model's input specifications, and fed into the CNN model for prediction. The predicted label with the highest probability is then displayed in real-time using OpenCV, providing seamless interaction and interpretation of ASL gestures.

Split dataset : Automating the dataset splitting process is crucial for effective model evaluation and validation. This script facilitates this by using a folder splitting library, defining input and output directories, and executing the split based on a specified ratio. By dividing the dataset into training and validation sets, it ensures a robust evaluation of the trained model's performance.

Train the model : The building, training, and evaluation of the CNN model.

Model architecture can be defined as a sequential model, consisting of convolutional layers with increasing filter sizes, dropout layers for normalization, and dense layers for classification.

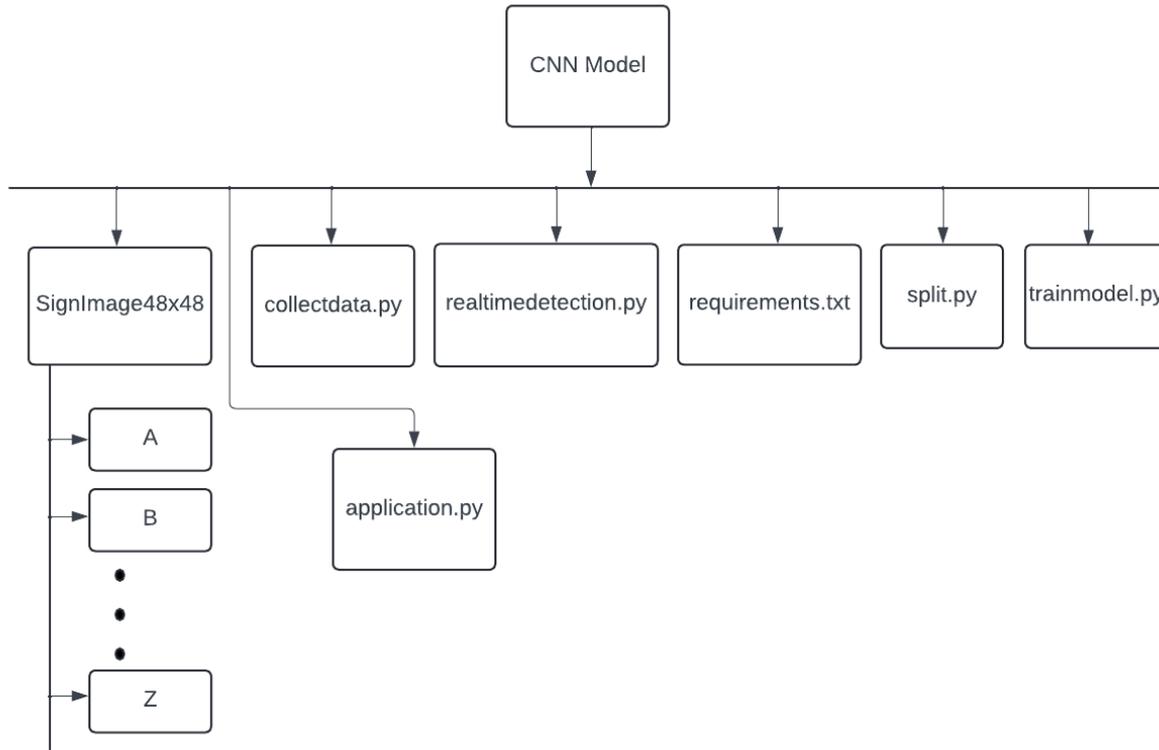


figure 2 : CNN Model – Script Hierarchy

Working of LSTM Model

The sign language detection model, using the Long Short-Term Memory (LSTM) architecture, facilitates real-time recognition of sign language gestures. Its comprehensive framework includes several essential components, meticulously designed to ensure efficient collection, pre-processing, model training, and real-time detection.

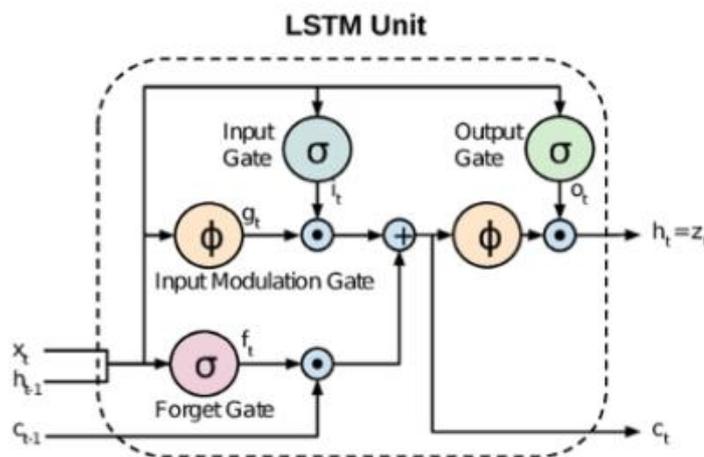


figure 3 : Long Short-Term Memory (LSTM) [8]

Utility Functions : The utility functions module encapsulates essential functions like image processing with MediaPipe, landmark drawing, and keypoint extraction. It also defines crucial global variables such

as data paths, actions, sequence counts, and lengths, providing a structured foundation for subsequent processes.

Collect data : Responsible for collecting training data, the collect data script captures video frames from the webcam, detects hand landmarks using MediaPipe, and saves them as images corresponding to various sign language gestures from A to Z, ensuring a rich and diverse dataset for robust model training.

Data Processing : In the data processing module, collected data undergoes careful processing, with keypoints extracted from detected palm landmarks and saved as NumPy arrays. Each image sequence for a specific gesture is meticulously organized and labeled into separate folders, ensuring data integrity and accessibility for subsequent steps.

Train the model : The train model script orchestrates the training of a sign language detection model using processed data, employing LSTM-based architecture for classification. The model configuration file, stored in JSON format, encapsulates crucial information about classes, configurations, activation functions, and initialization, facilitating transparent deployment and ensuring consistency across different executions.

Overall architecture: The project architecture seamlessly integrates the data collection, preprocessing, model training, and real-time detection phases. Starting with data collection and pre-processing, it proceeds to robust model training, creating a real-time application capable of accurately recognizing sign language gestures.

Through meticulous design and implementation, each component works together to provide a cohesive and effective solution, poised to make a significant contribution to the interpretation and accessibility of sign language.

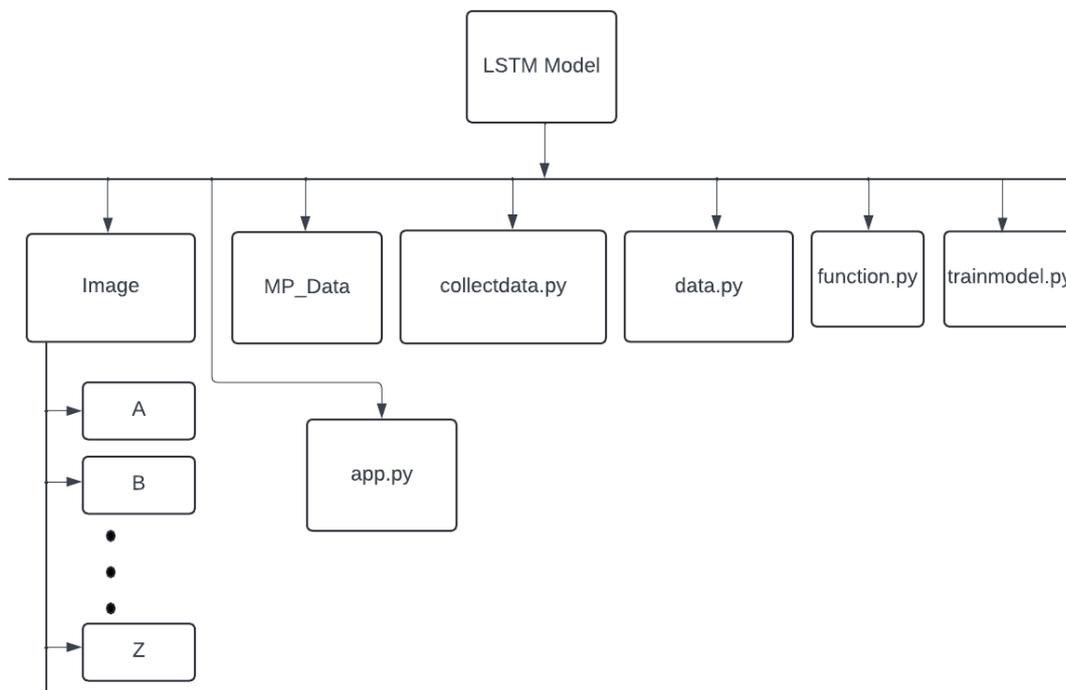


figure 4 : LSTM Model : script hierarchy

Compilation is done with the adam optimizer, categorical crossentropy loss function, and accuracy metrics, followed by training using the fitting method with the specified data generators and training parameters.

FUNCTIONS AND METRICS USED

SoftMax Function:

The SoftMax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

σ = softmax

\vec{z} = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector

figure 5 : Softmax Function

In both CNN and LSTM models for sign language detection, the SoftMax function is used in the output layer to convert raw output scores into probabilities.

Adam Optimizer:

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm.

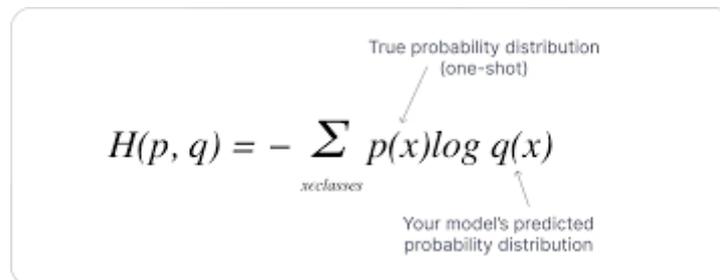
$$\begin{aligned} \nu_t &= \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t \\ s_t &= \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \\ \Delta\omega_t &= -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t \\ \omega_{t+1} &= \omega_t + \Delta\omega_t \end{aligned}$$

η : Initial Learning rate
 g_t : Gradient at time t along ω^j
 ν_t : Exponential Average of gradients along ω_j
 s_t : Exponential Average of squares of gradients along ω_j
 β_1, β_2 : Hyperparameters

figure 6 : Adam Optimizer [9]

Categorical Cross-entropy loss (SoftMax loss):

The cross-entropy loss function measures your model’s performance by transforming its variables into real numbers, thereby evaluating the 'loss' associated with them. The higher the difference between the two, the higher the loss. The cross-entropy loss function measures your model’s performance by transforming its variables into real numbers, thereby evaluating the 'loss' associated with them. The higher the difference between the two, the higher the loss.



The diagram shows the formula $H(p, q) = - \sum_{x \in \text{classes}} p(x) \log q(x)$ inside a rounded rectangle. An arrow points from the text 'True probability distribution (one-shot)' to the $p(x)$ term. Another arrow points from the text 'Your model's predicted probability distribution' to the $q(x)$ term.

figure 7 : Categorical Cross-entropy Loss [10]

Accuracy Metric:

Accuracy is a metric that measures how often a machine learning model correctly predicts the outcome. You can calculate accuracy by dividing the number of correct predictions by the total number of predictions. In other words, accuracy answers the question: how often the model is right?

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error

n = number of data points

Y_i = observed values

\hat{Y}_i = predicted values

figure 8 : Accuracy Metric (Mean Squared Error) [11]

4. Comparative Analysis: CNN vs LSTM

a. Data Collection and Preparation

- CNN Model:

Data Collection: Image of Sign Language gestures is recorded using a webcam. Each gesture is associated with a specific hand sign that corresponds to a letter of the alphabet. The webcam captured a live video image and the user could trigger the capture by performing a specific gesture associated with a letter. These captured images are then saved to folders in the local file system, with each folder representing a different class. For example, if the user makes a gesture corresponding to the letter "A", the webcam will capture an image and this image will be saved in a folder titled "A".

Data preparation: After image collection, the dataset is divided into training and validation sets. This ensures that the model can learn from a portion of the data and evaluate its performance on unseen data. The Split.py script is used to split the dataset into training and validation sets, possibly based on a predefined ratio (80% for training, 20% for validation). Each image is pre-processed and resized to a fixed size (48 x 48 pixels) to ensure uniformity across the entire dataset.

- LSTM Model:

Data Collection: Skeletal data series representing human gestures collected using depth sensors. These sensors collect information about the position of different body parts over time. Each sequence represents a specific gesture performed by the user, with each frame in the sequence containing skeletal data recorded at a specific time. Bone data may include information about the position and movement of key joints in the user's body, such as the hands, arms, and head.

Data preparation: After collecting the bone data series, the data set is divided into training and testing sets (80,20). This decomposition makes it possible to train an LSTM model on a portion of the data and evaluate its performance on invisible sequences.

b. Model Architecture

- CNN Architecture:

- **Convolutional layer:** CNN architecture starts with one or more convolutional layers, which combine input images with learnable filters to extract spatial features. These filters capture patterns at different spatial scales in the image.

- **Max Pooling Layer:** After the convolution layer, the max pooling layer is used to sample feature maps, thus reducing their spatial dimension while retaining the most important information. Max pooling makes the model more robust to input image variations and reduces computational complexity.

- **Dropout layers:** Dropout layers are added after convolutional layers to avoid overfitting. Dropout randomly removes a portion of neurons during training, forcing the network to learn more robust and general features.

- **Dense layer:** After the convolution layer and max pooling layer, one or more dense (fully connected) layers are added for classification. These layers integrate the spatial features extracted by the convolutional layers and learn to map them to the output layers. The final dense layer uses a SoftMax activation function to generate class probabilities.

- LSTM Architecture:

- **LSTM layer:** Unlike CNNs, which are designed to process spatial data such as images, LSTM networks are specialized for sequential data. LSTM architecture consists of one or more LSTM layers, which are variants of recurrent neural networks (RNN) capable of capturing long-term dependencies in sequential data.

- **Dense layer:** Similar to CNN architecture, dense layer is used for classification after LSTM layer. These layers take the output of the LSTM layers and map it to the output layers. The final dense layer uses a SoftMax activation function for multi-class classification tasks.

c. Model Configuration:

- CNN Configuration:

- **Convolutional layers:**

- Four convolutional layers with increasing filter sizes (128, 256, 512, 512).
- Each followed by max pooling (2x2) and dropout (0.4).
- Parameters: Total trainable parameters are 4183174.

- **Dense layers:**

- Multiple fully connected dense layers (512, 64, 256, 64, 256).

- Each layer is followed by an abort (0.4 for most layers, 0.2 for some layers).
- Output layer: Six SoftMax enabled neurons.

- **Training setup:**

- Adam optimizer, categorical cross-entropy loss, precision metric.
- Trained for 100 epochs.

- LSTM Configuration:

- **LSTM layer:**

- Three LSTM layers (64, 128, 64 units).
- Final LSTM layer, followed by dense layers (64, 32) and SoftMax output.
- Parameters: Total trainable parameters are 188090.

- **Training setup:**

- Adam optimizer, classification cross-entropy loss, measure of classification accuracy.
- Trained for 100 epochs.

d. Model Accuracy and Loss:

In terms of performance evaluation, training accuracy and validation accuracy were calculated for each model.

Accuracy:

Training accuracy is the percentage of correct predictions the model makes using the training data. It is calculated by comparing the model predictions on training data with the actual values of the training data.

High training accuracy may indicate that the model learned the training data well, but it does not necessarily mean the model will perform well on unseen data.

Validation accuracy is the percentage of correct predictions the model makes on the validation data. Validation data is a subset of training data that the model did not see during training.

High validation accuracy shows that the model generalizes well to unseen data.

In our case, training and validation accuracy increases over many epochs for both the models, which is a good sign. This indicates that our model learns from training data and can generalize well to data not seen in the validation set.

Loss:

Training loss measures how well the model fits the training data. It is calculated by summing the errors. The training loss gradually decreases over the epochs indicating that the model learns effectively from the training data.

On the other hand, **Validation loss** evaluates the model's performance on the validation set, which is the part of the dataset dedicated to validating the model's performance.

The reduced validation loss implies that the model generalizes well to unseen data and does not just remember the training data.

As training and validation losses decrease over epochs, this shows that both models are learning and improving their performance on the training and validation datasets.

Accuracy Evolution:

“Accuracy evolution” refers to the change in model accuracy over time, typically over multiple epochs during training.

It is displayed using a line graph, where the x-axis represents the number of epochs (or time) and the y-axis represents the precision.

Monitoring the evolution of accuracy can provide valuable information about the model learning process.

If accuracy on the training set continuously improves (i.e. error decreases), but accuracy on the validation set begins to deteriorate (i.e. assuming error is increasing), then this is an indication that the model may be overfitting the training data.

Conversely, if the accuracy during training and validation improves over time, this indicates that the model learns effectively and generalizes well to unseen data.

However, if the accuracy does not change or begins to decrease, which may indicate the model has stopped learning or is starting to forget previously learned information.

Below are the observed graphs for accuracy and loss evolution for CNN and LSTM model:

CNN Accuracy and Loss Evolution Graph:

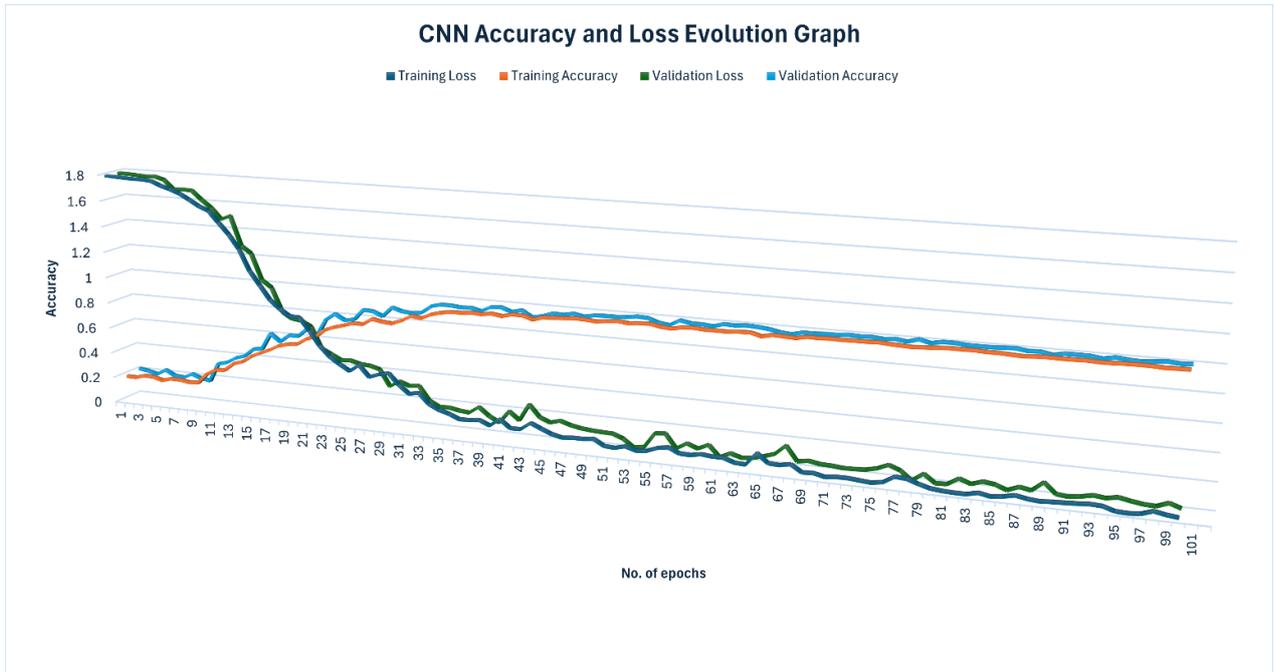


figure 9 : CNN Accuracy and Loss Evolution Graph

LSTM Accuracy and Loss Evolution Graph:

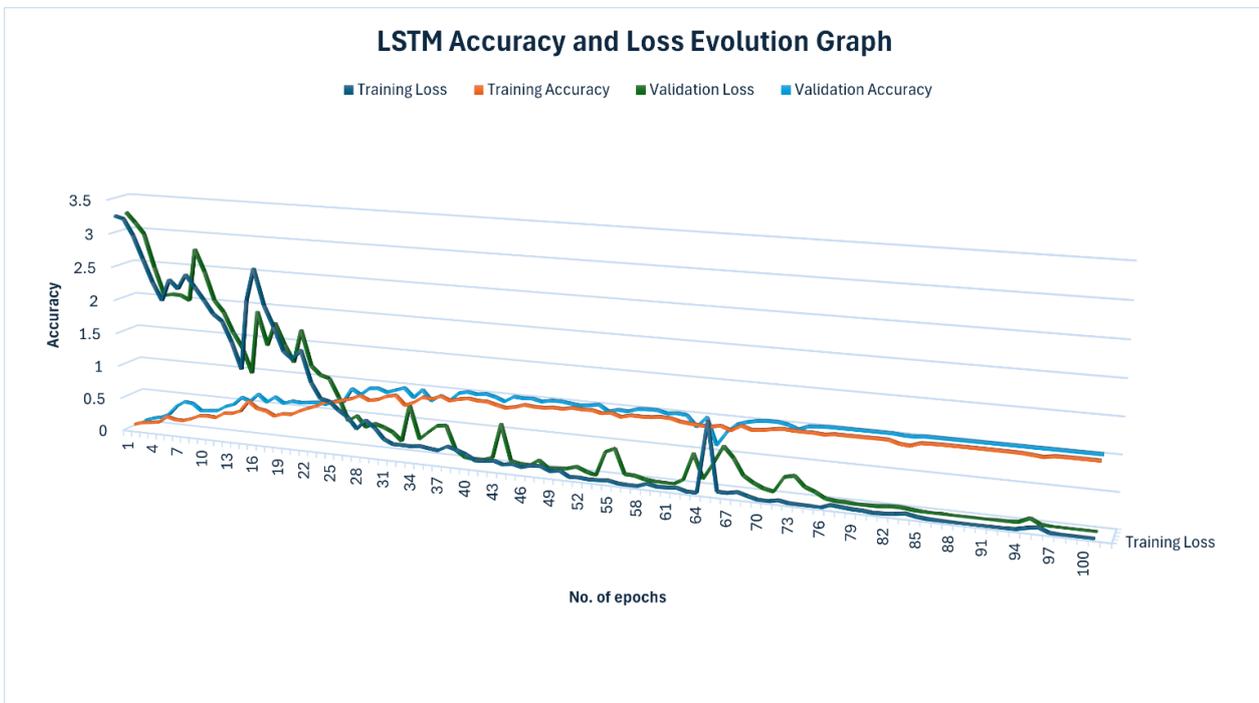


figure 10 : LSTM Accuracy and Loss Evolution Graph

CNN model achieved **99.48%** accuracy and the LSTM model achieved **99.92%** accuracy, which both are promising.

e. Training time:

- CNN training time:

CNNs typically have faster training times than LSTMs. This is because:

- **Parallel Processing:** CNN processes input data in parallel. They simultaneously apply convolution filters across the entire image, allowing for efficient computation.
- **Local Receptive Fields:** CNN focuses on local receptive fields (small regions of the input). This local processing reduces the overall computational load.
- **Weight sharing:** CNN shares weights between spatial locations, further accelerating the training process.
- **GPU Acceleration:** Modern deep learning library optimizes CNN calculations using GPU, thereby improving speed.

- LSTM training time:

LSTM involves sequential processing and takes longer. This is because:

- **Sequential nature:** LSTM processes input sequences step by step. Each time step depends on the previous step, introducing sequential dependencies.
- **Backpropagation over time (BPTT):** During training, LSTM performs BPTT, which includes calculating gradients over the entire sequence. This can be computationally expensive.
- **Complex Cell State Update:** LSTM maintains cell (memory) and gate (input, forget, output) state. Updating these states requires additional computation.
- **Limited Parallelization:** Due to sequential dependencies, LSTM cannot be fully parallelized over multiple time steps.

Therefore, if training time is an important factor, CNNs are preferable due to their parallelism and efficient weight sharing. However, LSTM is still useful for tasks where sequential context is important.

f. Performance:

- CNN for image-related tasks:

- **Strengths:**

- Spatial hierarchy: CNN excels at capturing spatial hierarchies in images. They learn the characteristics of raw pixels, detecting edges, textures, and patterns.
 - Robust against variations: CNN handles variations in lighting, background, and object orientation.
 - Image Classification: They are widely used for image classification, object detection and segmentation.
- **Limitations:**
 - Missing temporal context: CNN processes each frame independently. They inherently do not model temporal dependence.

- LSTM for Sequential Data:

- **Strengths:**
 - Temporal Modelling: LSTM is designed for sequential data. They capture long-term dependencies and contexts.
 - Natural Language Processing (NLP): LSTM excels at NLP tasks where word order is important.
 - Sign Language Sequence: LSTM is suitable for sign language recognition due to the temporary nature of gestures.
- **Limitations:**
 - Complexity: LSTM has more parameters and requires longer training time.
 - Overfitting: If not properly normalized, LSTM can overfit on small data sets.

5. Comparison Table: CNN vs LSTM

The above comparative analysis can be summarized as follows:

Basis of Differentiation	CNN Model	LSTM Model
Data Collection	Images of sign language gestures were captured using a webcam. Each gesture was associated with a specific hand sign corresponding to a letter of the alphabet.	Sequences of skeletal data representing human gestures were collected using depth sensors. These sensors capture information about the positions of various body parts over time.
Data Preparation	After collecting the images, the dataset was split into training and validation sets (80,20). Each image was pre-processed and resized to a fixed size (48x48 pixels) to ensure uniformity across the dataset.	After collecting the skeletal data sequences, the dataset was split into training and testing sets (80,20).
Data Modality	The project deals with image data captured by a webcam.	The project deals with sequential skeletal data captured by depth sensors.

Representation	Images provide spatial information about the hand gestures.	Skeletal data captures temporal dynamics and movement patterns.
Preprocessing	Images require resizing and normalization.	Skeletal data require normalization and feature engineering.
Model Input	CNNs process 2D image data directly.	LSTMs handle sequential data, such as time-series or sequences of feature vectors.
Architecture	The CNN architecture starts with one or more convolutional layers, followed by max-pooling layers, dropout layers, and dense layers for classification.	The LSTM architecture consists of one or more LSTM layers, which are recurrent neural network (RNN) variants capable of capturing long-term dependencies in sequential data. Dense layers are used for classification after the LSTM layers.
Feature Extraction	CNNs use convolutional and max-pooling layers to extract spatial features hierarchically from input images.	LSTMs capture temporal dependencies within sequential data using recurrent connections.
Model Complexity	CNNs are simpler in structure compared to LSTMs.	LSTMs can capture complex temporal patterns and dependencies in sequential data, which may be crucial for tasks like gesture recognition where the order of movements matters.
Configuration Details	The CNN project has four convolutional layers with increasing filter sizes, several fully connected dense layers, and six neurons with SoftMax activation in the output layer. It uses the Adam optimizer, categorical cross-entropy loss, and accuracy metric. It was trained for 100 epochs.	The LSTM project has three LSTM layers, followed by dense layers and SoftMax output. It uses the Adam optimizer, categorical cross-entropy loss, and categorical accuracy metric. It was trained for 100 epochs.
Data Type	The project uses image data.	The project uses sequential skeletal data.
Model Complexity	Has more parameters due to convolutional layers.	Fewer parameters compared to CNN but requires longer training times due to sequential processing.
Training Time	Faster training times compared to LSTM.	Slower training times due to sequential processing.
Performance	Performance heavily depends on the nature of the data and the complexity of the task. CNNs are more commonly used for image-related tasks.	Performance heavily depends on the nature of the data and the complexity of the task. LSTMs are preferred for sequential data such as time series, natural language processing or sign language detection.
Accuracy	99.48	99.92

6. Conclusion:

We have obtained important insights into the properties and performance of CNN and LSTM models for sign language detection through our comparative analysis:

- We found that CNN and LSTM models differ significantly in data type, architecture, and model complexity through testing and analysis.
- While the LSTM model works on sequential data, interpreting each action as a time series of feature vectors, the CNN model analyses RGB images of sign language movements.
- The LSTM model consists of LSTM cells with input, forgetting, and output gates, but the CNN model consists of convolutional and max pooling layers.
- Additionally, our evaluation metrics, including training accuracy and validation accuracy, highlighted the model's ability to adapt to training data and generalize Unseen data. By comparing these metrics, we gained insight into model performance and identified potential issues such as overfitting or underfitting.
- We also plot the accuracy for both models, showing their performance over epochs. It is important to note that additional training and testing is required for both models.
- The reported accuracy, with the CNN model achieving an accuracy of 99.48% and the LSTM model achieving an accuracy of 99.92%, is very promising.
- However, it is essential to realize that these accuracies may not fully represent the capabilities of the models due to the small dataset used. This limitation is mainly due to hardware limitations of our local machine.
- Therefore, future research should focus on collecting larger datasets and conducting more in-depth experiments to better evaluate CNN and LSTM models for sign language detection tasks.
- Despite these limitations, our study provides valuable insights into the field of sign language detection and paves the way for future advances in this field.

7. References:

- [1] (PDF) Real-Time Sign Language Detection Using CNN - ResearchGate. https://www.researchgate.net/publication/364185120_Real-Time_Sign_Language_Detection_Using_CNN.
- [2] Sign Language Detection using LSTM - IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/10080705>
- [3] Sign Language Recognition Using Long Short-Term Memory Deep Learning Model. https://link.springer.com/chapter/10.1007/978-981-99-7093-3_46.
- [4] CNN and Stacked LSTM Model for Indian Sign Language Recognition. https://www.researchgate.net/publication/340431992_CNN_and_Stacked_LSTM_Model_for_Indian_Sign_Language_Recognition.
- [5] Machine learning based sign language recognition: a review and its

https://www.researchgate.net/publication/343955537_Machine_learning_based_sign_language_recogniti_on_a_review_and_its_research_frontier

[6] American Sign Language (ASL) Detection System using Machine Learning
<https://www.doi.org/10.55041/IJSREM27702>

[7] CNN Architecture Diagram

<https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>

[8] LSTM Architecture Diagram

<https://www.naukri.com/code360/library/bidirectional-lstm>

[9] Adam Optimizer

<https://discuss.pytorch.org/t/how-adam-optimizer-influence-the-learning-rate/168543>

[10] Cross-entropy

<https://www.v7labs.com/blog/cross-entropy-loss-guide>

[11] Accuracy Metric

<https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall>