WLASL recognition

Basel Elzatahry

April 10, 2023

Abstract

Sign language is one of the most significant tools to learn, yet one of the most under-looked languages when it comes to technological applications. We have all the tools necessary to learn all different spoken languages world wide; however, learning American Sign Language is very hard for people, especially when they are not in a motivating environment to learn the language. The goal of this thesis is to build a deep learning model that can accurately recognize and understand ASL gestures in real-time. Specifically, the model will focus on recognizing hand gestures, facial expressions, and body movements in sign language. The ultimate objective of this work is to break down communication barriers and promote inclusivity for the Deaf community. Which is why I decided to expand on the current research on the subject and utilize existing Deep Learning algorithms to get a step closer to a reliable solution. The results I have obtained show that CNN-LSTM networks are promising and can be trained over bigger Datasets than others, yielding better results and performance. There has been some challenges and limitations, but the results have confirmed the viability of the explored option with a big room of improvement given more time and tools.

1 Introduction

Like humans, deep learning models can be trained to recognize patterns in any sort of data that it is exposed to if provided a large amounts of labeled examples. This can be applicable to almost all kinds of data; visual, audible, or text. The way the model is trained to recognize these patterns is very similar to that of the human brain; during the training process, the model would learn to recognize patterns in the data that are associated with labels. Once trained, the model can be used to predict the label of new, unlabeled data.

A specific example that I have decided to work in is learning languages. The key to language learning is the ability to recognize patterns in languages and to use those patterns to generalize , understand, and create new language expressions. This process of recognizing and generalizing patterns is fundamental to both human learning and deep learning models. As humans, it takes us time to learn a new language. The process starts with us learning the small things in the language, like the alphabet. Afterwards, we learn how to connect letters together to form words, and then we learn how to connect these words and form sentences that are

comprehensible. During that process, we are being exposed to new data continuously. The more the data we are exposed to, the more fluent we are in the language. On a high-level, this is how constructed Deep Learning models work. Models are trained on huge Datasets with labeled inputs, so they are able to recognized unlabeled data; as mentioned earlier, the more data we have, the easier the model's recognizing of the patterns and the better its performance.

Overall, the similarities between how humans learn languages and how deep learning models work suggest that these models can be powerful tools for language learning and natural language processing. By exposing these models to large amounts of labeled and unstructured data, we can train them to recognize patterns in language data and generate new language expressions, which can be used for a variety of practical applications.

Accordingly, I have chosen to utilize Deep Learning tools to construct a model that can accurately recognize real-time word-level American Sign Language (WLASL). This has been made viable by introducing the constructed model to a sufficiently-big Dataset of labeled videos of American Sign Language words. It is important to understand that the Dataset is only videos of separate words without a context to how these words can be used to actually form sentence; this means that goal of this work is to construct a model that is only capable of translating words and cannot construct sentences. The model will focus on different features, like hand gestures, facial expressions, and body movements. These are all contributing factors that humans need when they learn a language, and so does the model. To optimize the performance of the chosen algorithms for our dataset, we have experimented with various hyperparameters and regularization techniques. Additionally, we have explored the potential benefits of data augmentation techniques, such as geometric transformations and random noise injection, to increase the size and diversity of our dataset.

Needless to say, the deaf community face a lot of struggles on daily basis and are continuously impacted by our ignorance of American Sign Language. There has been a lot of research done in the past on that problem; however, we are yet to find a reliable solution that solves the problem and breach the gap between the deaf community and the rest. Many of these solutions also utilized Deep Learning or other Machine Learning tools in an effort to create a robust model capable of understanding Word-level American Sign Language. There exists a few models that use Deep Learning and have promising results, which is why I decided to build up on these models and address some of the difficulties and limitations they face in a striving to take one more step forward towards the desired solution.

1.1 Objective

Through the application of deep learning techniques, this research aims to help in the development of an accurate and efficient system for recognizing American Sign Language (ASL) gestures. The study will involve the collection and analysis of a large dataset of ASL gestures, the development and training of deep learning models, and the evaluation of the performance of the system using various metrics. The results of this research will contribute to the advancement of technology for assisting individuals who are deaf or hard of hearing, and could have broader implications for

the fields of computer vision and natural language processing.

The research that has already been done on this problem in the past has proved to us that this problem can be solved using Deep Learning tools; however, we have not expanded on the subject enough to reach the solution. In my research, I'm focusing on developing existing solutions to improve the robustness of ASL recognition using deep learning techniques. In other words, the main problem I am addressing is the size of the dataset and the goal is to build a model that can learn as many words as possible without dropping its accuracy in comparison to existing solutions.

1.2 American Sign Language

Firstly, American Sign Language (ASL) is a visual language that uses a combination of hand gestures, facial expressions, and body language to communicate. It is the primary language used by the deaf community in the United States and Canada. While ASL is recognized as a distinct language, it is not widely understood or used by hearing individuals, which can create significant barriers for the deaf community. According to a report by the National Deaf Center, deaf people in the United States

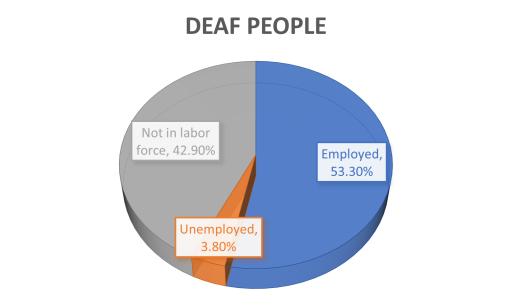

Figure 1: Employment for deaf people

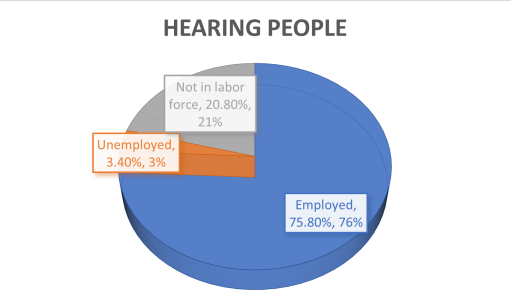

Figure 2: Employment for hearing people

face significant challenges when it comes to employment. The report found that deaf individuals are more likely to be unemployed or underemployed than their hearing peers, with only 53.8% of working-age deaf individuals employed compared to 75.8% of their hearing counterparts. One of the primary reasons for this disparity is the lack of access to education and training opportunities.

The lack of ASL knowledge among hearing individuals is a significant barrier to employment for deaf individuals. Many employers are hesitant to hire deaf individuals because they are unsure how to communicate with them, or they assume that it will be too difficult or expensive to provide accommodations. This can lead to a lack of job opportunities and lower pay for deaf individuals, even when they have the same skills and qualifications as their hearing peers.

In addition to employment barriers, the lack of ASL knowledge can also impact deaf individuals' access to healthcare. A study by Boston University found that deaf patients are more likely to experience communication barriers and receive lower-quality care than their hearing peers. The study found that many healthcare providers do not have access to ASL interpreters or other communication tools, which can lead to misunderstandings, misdiagnoses, and inadequate treatment.

The lack of ASL knowledge among hearing individuals can also lead to social isolation and discrimination for deaf individuals. Without access to a common language, deaf individuals may struggle to communicate with hearing friends and family members, participate in community events, or access public services. This can lead to feelings of isolation, loneliness, and frustration, and can limit their ability to fully participate in society.

Despite these challenges, the deaf community continues to advocate for their rights and the recognition of ASL as a distinct language. In a recent report, Human Rights Watch called for governments to recognize sign languages as official languages and provide access to ASL interpreters in all public services. The report also highlighted the importance of providing education and training opportunities for deaf individuals to help overcome barriers to employment and other areas of life.

Overall, the lack of ASL knowledge among hearing individuals creates significant challenges and barriers for the deaf community. To improve access to education, employment, healthcare, and other services for deaf individuals, it is essential to capitalize on all technologies available to us and improve it to break all barriers. This solution is not to be found overnight, but we can only take small steps to find it, and this research serves as one of those steps.

1.3 Dataset

The quality and relevance of the dataset used is crucial to the success of any data based project. Therefore, it's essential to carefully consider the choice of dataset to ensure it aligns with our goals and research questions. So, let's now take a closer look at the dataset and the insights it can offer. In this project we havexve been working on 2 different datasets which were reasonably different from each other. The main reason for that is to make my model as broad as possible and to give it a chance to work in different environments so it would be more general. In this section I will be explaining both datasets in details, so we have a good understanding of what they look like and how trainable they are and then accordingly have a reasonable goal for the performance of our model.

1.3.1 First Dataset

This Dataset is a collection of American Sign Language (ASL) gestures recorded using a data glove. The data glove is a wearable device that captures the hand movements of the signer in 3D space. The dataset includes video recordings of 250 ASL gestures performed by 112 signers, with each gesture recorded 20 times for a total of 5000 samples.

The dataset contains a wide range of ASL gestures, including both simple and complex signs. Each video recording shows a signer performing a single ASL gesture against a plain background. The signer is positioned such that their hand movements can be clearly seen, and the recordings are made under controlled lighting conditions to ensure consistency.

Each gesture in the dataset is labeled with its corresponding ASL gloss and English translation. The ASL gloss is a word-by-word transliteration of the ASL sign, while the English translation provides the equivalent meaning in spoken English.

The dataset also includes information about the signer, such as their gender, age, and hand dominance, which can be useful for studying individual variation in sign language.

The videos in the dataset are stored in AVI format, with each video file corresponding to a single ASL gesture. The videos have a resolution of 320x240 pixels and a frame rate of 30 frames per second. In addition to the video data, the dataset also includes 3D joint angle data for each gesture, which describes the angles between the fingers, hand, and wrist during the sign. This data can be used to study the kinematics of ASL gestures and to develop models that can recognize ASL signs based on hand movements.

Overall, this dataset can be used to develop and evaluate deep learning models for ASL recognition, as well as to investigate individual variation in sign language and the kinematics of hand movements in ASL.

1.3.2 Dataset 2

The WLASL (Word-level American Sign Language) dataset is a large-scale dataset of American Sign Language (ASL) videos, consisting of over 1,000 signs from the ASL lexicon. The dataset contains over 20,000 annotated videos, each showing a single sign, with an average duration of about 5 seconds. The videos were collected from the internet and span a wide variety of signing styles and camera viewpoints.

Each video is labeled with its corresponding sign, as well as additional metadata, such as the gender and age of the signer, the presence of gloves, and the camera angle. The signs are categorized into 2,000 unique glosses, which are English glosses representing the meaning of the sign. The dataset also includes precomputed optical flow, a useful feature for video analysis, as well as the ability to download the videos directly from the repository.

The WLASL dataset is an important resource for the development and evaluation of machine learning algorithms for ASL recognition. It provides a large, diverse, and well-annotated collection of videos, which can be used to train and test deep learning models for ASL recognition at the word level. The dataset has already been used in a number of research studies, demonstrating its usefulness in advancing the state-of-the-art in ASL recognition. The availability of the dataset has also led to a growing interest in the development of technologies to improve communication and accessibility for the Deaf community.

2 Background

As mentioned, he research presented in this thesis aims to develop an accurate and efficient system for real-time American Sign Language (ASL) recognition by utilizing deep learning techniques. Although previous research has explored various deep learning algorithms, our study focuses on experimenting with and combining these algorithms to improve their performance on our dataset.

Our research utilizes several state-of-the-art deep learning algorithms, including Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and transformer models. CNNs have proven to be highly effective in image

recognition tasks, while LSTM networks can capture temporal dependencies in sequential data such as sign language gestures. Transformer models, on the other hand, have shown great success in natural language processing tasks and may be useful in capturing the semantic meaning of sign language gestures.

In this section, we will breakdown all the algorithms we used, explain our dataset in detail, discuss our different performance metrics, and show some of the previous work that we expanded on.3

2.1 Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI) that involves the development of algorithms and statistical models that enable computer systems to automatically learn from and make predictions or decisions based on data, without being explicitly programmed for each decision or prediction. In this detailed explanation of machine learning, we will cover the various components and processes involved in machine learning, as well as some of its applications and challenges.

In machine learning, there are several types of learning, including supervised, unsupervised, and reinforcement learning. While all three are relevant to the field, for the purposes of this research, we will focus on supervised and unsupervised learning.

Supervised learning is a widely-used machine learning technique that involves training a model to predict an output variable based on one or more input variables, using a labeled dataset. The input variables are also known as predictors or independent variables, while the output variable is known as the target or dependent variable.

In supervised learning, the labeled dataset is used to teach the model how to make predictions by minimizing the difference between its predicted outputs and the true outputs in the training data. This is achieved through an optimization process, where the model iteratively adjusts its parameters to minimize a loss function that measures the difference between its predicted outputs and the true outputs in the training data.

Supervised learning can be used for both classification and regression tasks. In classification tasks, the goal is to predict discrete categories, while in regression tasks, the goal is to predict continuous values. Some common examples of supervised learning applications include predicting whether a customer will purchase a product, diagnosing a medical condition based on patient symptoms, and predicting the price of a house based on its features.

Supervised learning is particularly useful in scenarios where we have a clear idea of what we want the model to predict, and we have labeled examples to train it on. It has been applied successfully in a wide range of fields, including finance, healthcare, and marketing. However, one of the main challenges of supervised learning is the availability and quality of labeled data, which can be time-consuming and costly to obtain.

Figure 3 shows a visual representation of the process of supervised learning. The labeled dataset is divided into a training set, which is used to train the model, and a validation set, which is used to tune the model's hyperparameters and evaluate its performance. Once the model is trained and validated, it can be tested on a separate test set to estimate its performance on new, unseen data.

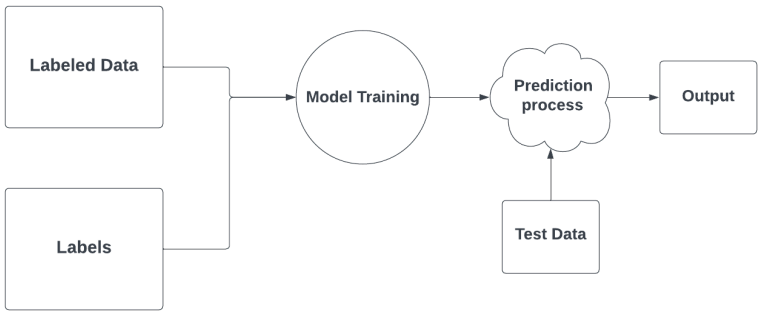

Figure 3: Supervised learning

In summary, supervised learning is a powerful machine learning technique that involves training a model to predict an output variable based on one or more input variables, using a labeled dataset. It has many practical applications, but its effectiveness depends heavily on the availability and quality of labeled data.

In contrast, unsupervised learning is a machine learning technique that involves training a model to identify patterns or structure in a dataset without any labeled output variables. The goal of unsupervised learning is to find patterns or relationships in the data that can help us better understand the data or make predictions about new data.

Unsupervised learning algorithms work by exploring the structure of the data and identifying clusters, patterns, or relationships that exist within it. Clustering is a common example of unsupervised learning, where the goal is to group similar data points together based on their characteristics.

Unlike supervised learning, unsupervised learning does not have a predefined objective or target variable. Instead, it relies on the model to identify meaningful patterns or relationships in the data that may not be immediately apparent to humans. This makes unsupervised learning particularly useful when we don't have labeled examples, or when we want to explore the structure of the data without any preconceived notions about what we might find.

There are several types of unsupervised learning algorithms, including clustering, dimensionality reduction, and anomaly detection. Clustering algorithms group data points together based on their similarity, while dimensionality reduction algorithms simplify the data by identifying the most important features or components. Anomaly detection algorithms identify data points that are significantly different from the rest of the dataset.

Unsupervised learning has many practical applications, such as identifying customer segments based on their behavior, detecting fraud in financial transactions, or clustering images to identify common themes. However, it can be challenging to evaluate the performance of unsupervised learning algorithms, as there is no predefined objective or target variable to measure against.

Figure 4 shows a visual representation of how unsupervised learning works. Unlike supervised learning, unsupervised learning does not have a clear objective or target variable, and the model must identify meaningful patterns or relationships in the

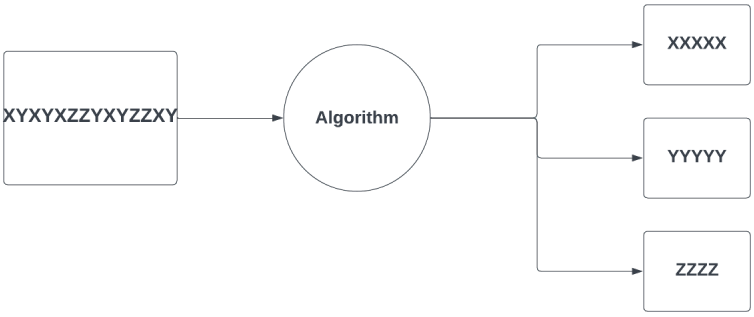

Figure 4: Unsupervised learning

data without any labeled examples. The output of an unsupervised learning algorithm is typically a set of clusters or patterns that can be used to better understand the structure of the data or make predictions about new data.

Now that we have discussed the different types of machine learning, it's important to understand the machine learning pipeline, which consists of several steps involved in building a successful machine learning model. Each step is essential in ensuring that the model can accurately learn from the data and provide reliable predictions. Figure 5 shows the machine learning process pipeline.

The machine learning pipeline begins with data ingestion, where data is collected

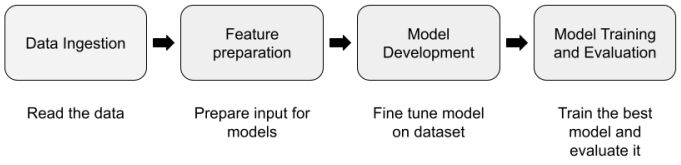

Figure 5: Ml pipeline

and prepared for use in the model. Data cleaning, formatting, and preprocessing are necessary in this step to ensure that the data is suitable for analysis. The quality of data is crucial to the accuracy of the model, as it directly affects the model's ability to make predictions.

After data ingestion, the next step is feature preparation, where relevant features are selected and transformed to be used in the model. Feature selection involves

choosing the most important features that will contribute to the model's predictive power while discarding irrelevant ones. Feature transformation is a critical process that involves converting the selected features into a format suitable for the model.

In the model development step, a suitable model is selected and developed based on the problem being solved and the data available. This is where knowledge of different machine learning algorithms and techniques comes into play. A model can be a simple linear regression or a complex deep learning model, depending on the problem being solved and the data available. Model development involves defining the architecture of the model, tuning hyperparameters, and optimizing its performance.

Finally, the model is trained and evaluated on the data to assess its performance and refine it as necessary. In the training and evaluation step, the model's performance is measured against a set of predefined metrics, such as accuracy, precision, recall, and F1 score. In this section, we will explain all the relevant performance metrics and the guidelines on how to choose the right one. This step is usually iterative, and multiple rounds of training and evaluation may be required to refine the model and improve its performance.

In conclusion, the machine learning pipeline is a critical process in building a successful machine learning model. Each step in the pipeline is essential in ensuring that the model can accurately learn from the data and provide reliable predictions. Understanding the machine learning pipeline is crucial for data scientists and machine learning engineers to develop robust and effective machine learning models.

Machine learning has numerous applications in a wide range of industries, including healthcare, finance, marketing, and more. Some common applications of machine learning include image and speech recognition, natural language processing, fraud detection, recommendation systems, and autonomous vehicles.

But as with any powerful technology, there are also potential risks and challenges associated with the use of machine learning. One major challenge is the potential for bias in the data and resulting models. If the training data is biased, for example, if it does not represent the full diversity of the population or if it contains discriminatory patterns, the resulting model may also be biased. This can lead to unfair or discriminatory outcomes, particularly in areas such as hiring or lending. A study by Buolamwini and Gebru (2018) found that facial recognition technology is less accurate for darker-skinned individuals, potentially leading to biased decisions in law enforcement and other domains. The problem of bias is not easily solved, particularly given the limitations of available data. However, steps such as carefully choosing representative datasets, increasing diversity in data collection, and including ethical considerations in the design process can help to mitigate these risks. In this paper, we will explore some of these solutions in more detail.

Machine learning is a rapidly evolving field, with new algorithms and techniques being developed all the time. One area of current interest is deep learning, which involves the use of neural networks with multiple layers to learn complex patterns in the data. Deep learning has achieved impressive results in areas such as image and speech recognition, but it also requires large amounts of data and computing power. Based on that, we have decided to focus more on deep learning algorithms to tackle this problem.

2.2 Deep Learning

Deep learning is a rapidly growing subfield of machine learning that has garnered significant attention due to its exceptional ability to address intricate challenges across multiple domains, including natural language processing, speech recognition, and image recognition. In this section, we will delve into the fundamental principles of deep learning and its working mechanisms, beginning with the basics. The figure presented in 6 illustrates the interrelationship between Deep Learning, Machine Learning, and other fields within the domain of computer science.

Deep learning has revolutionized many fields, including computer vision, natural

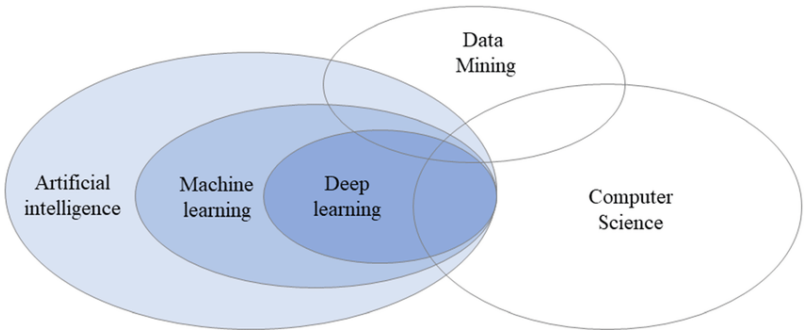

Figure 6: Relation between Deep Learning and Machine Learning

language processing, and speech recognition, and has enabled breakthroughs in tasks such as image and speech recognition, language translation, and game playing.

At a high level, deep learning models are composed of multiple layers of interconnected artificial neurons. Each neuron takes in one or more input values, applies a mathematical operation to them, and outputs a value that is transmitted to the next layer. The connections between neurons are weighted, meaning that the strength of the connection determines the influence of the output of one neuron on the input of another.

The deep learning model is trained using a large dataset of labeled examples. The model is initialized with random weights, and the training process adjusts the weights to minimize the difference between the model's output and the correct output, using a loss function that measures the error of the model's predictions. This is done using a technique called backpropagation, which calculates the gradient of the loss function with respect to each weight and updates the weights to move in the direction that reduces the loss.

One of the challenges in traditional machine learning approaches is that it requires manual feature engineering, where domain experts need to select and extract features that are relevant to the problem at hand. In contrast, deep learning models can automatically learn relevant features from raw data without the need for manual feature engineering. This is achieved by introducing non-linearities between layers of neurons in a neural network, which allows the model to learn complex representations of the input data. By automatically learning relevant features from raw data, deep learning models can potentially improve performance and reduce the time and effort required for feature engineering.

Deep learning models utilize optimization algorithms to train the neural network,

which involves minimizing the loss function by adjusting the weights and biases of the neural network. Stochastic gradient descent (SGD) is one of the widely used optimization algorithms in deep learning, which updates the weights after each training example, making it computationally efficient for large datasets. However, SGD can lead to slow convergence and difficulties in optimizing certain types of neural networks. As such, variants of SGD, such as Adam and RMSProp, have been developed to improve the optimization process by adapting the learning rate based on the history of the gradients, leading to faster convergence and better performance.

Another important technique used in deep learning is batch normalization, which normalizes the activations of each layer. The normalization process involves adjusting the mean and variance of the activations to reduce the internal covariate shift, which can help improve the performance and stability of deep learning models. Additionally, batch normalization allows for the use of higher learning rates, leading to faster convergence and better generalization.

Overall, optimization algorithms such as SGD, Adam, and RMSProp, along with techniques like batch normalization, play a crucial role in training deep learning models and improving their performance.

There are many types of deep learning models, each suited to different types of data and tasks. One of the most commonly used types is the feedforward neural network, which consists of multiple layers of neurons arranged in a sequence. Feedforward neural networks are used for tasks such as image classification, speech recognition, and natural language processing.

Convolutional Neural Networks

As we move towards a more detailed discussion of deep learning models, it is important to understand Convolutional Neural Networks (CNNs), a specialized type of neural network architecture that has proven highly effective for analyzing image and video data due to their ability to learn spatial features. In contrast to traditional neural networks, CNNs use a series of layers that extract and encode the features of an image in a hierarchical manner. This allows CNNs to learn complex features and relationships between pixels, which is essential for tasks such as image classification and object detection. Figure 7 shows and illustration of CNNs that is necessary to get a better understanding of them.

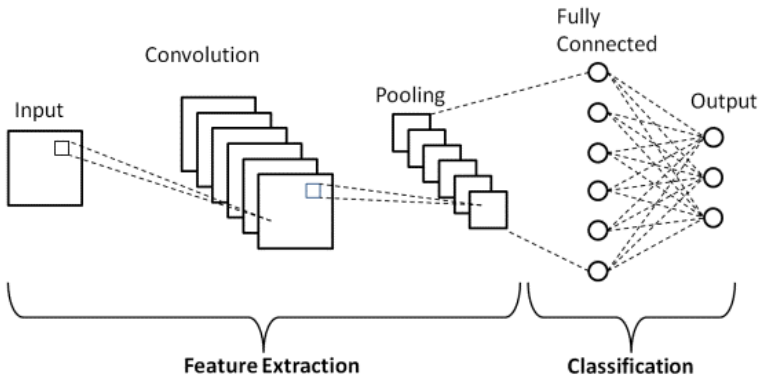

Figure 7: CNN illustration

Convolutional neural networks (CNNs) are a specialized type of neural network architecture that has proven highly effective for analyzing image and video data due to their ability to learn spatial features. CNNs are composed of a series of layers that extract and encode features of the input image in a hierarchical manner.

The first layer of a CNN is usually a convolutional layer, where learnable filters are convolved with the input image to produce feature maps. These filters are trained to identify important spatial features in the input image such as edges or corners. The output of the convolutional layer is then passed through a nonlinear activation function such as ReLU 1 to enable the network to learn more complex features. Pooling layers are also used in CNNs to reduce the spatial dimensionality of the data while retaining important features, which can help prevent overfitting and reduce computational costs. All these concepts will be explained in more details in the following sections.

$$Relu(z) = max(0, z) \tag{1}$$

A key advantage of CNNs is their ability to learn features that are translationally invariant, meaning they can recognize the same feature regardless of its location in the image. This is accomplished by using filters that are applied across the entire image, rather than being limited to a small region. This is important because in natural images, objects can appear at different locations in the image and a filter that is applied only to a small region will not be able to recognize the object if it appears in a different region.

Recent advances in CNN architecture have led to the development of deep CNNs, which consist of many layers of convolutional and pooling layers. These deep architectures have been shown to achieve state-of-the-art performance on a variety of image and video recognition tasks, including image classification, object detection, and segmentation.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of artificial neural networks that are capable of processing sequential data, such as time series or natural language data. RNNs have a unique architecture that allows them to maintain an internal state by using feedback connections between neurons. This architecture enables RNNs to process sequences of variable length, making them well-suited for many applications in which sequential data is involved. Please refer to figure 8 to get a general understanding of the structure of an RNN.

The basic building block of an RNN is the recurrent unit, which is responsible for maintaining and updating the network's internal state as it processes each element of the input sequence. In a simple RNN, the recurrent unit takes as input the current input element and the previous hidden state, and produces a new hidden state that is passed on to the next time step. However, simple RNNs can suffer from the vanishing gradient problem, which makes it difficult to learn long-term dependencies in sequential data.

To address this issue, Long Short-Term Memory (LSTM) networks were developed. LSTMs are a type of RNN that are specifically designed to capture long-term dependencies in sequential data. LSTMs achieve this by introducing a memory cell, which allows the network to selectively forget or remember information over long time

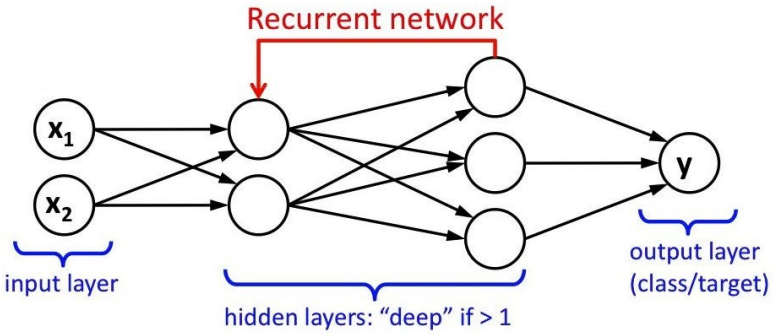

Figure 8: RNN illustration

intervals. This is accomplished through a set of gating mechanisms that control the flow of information into and out of the memory cell.

The three main gates of an LSTM are the input gate, the forget gate, and the output gate. The input gate controls how much new information should be stored in the memory cell, while the forget gate controls how much information from the previous state should be discarded. The output gate controls how much information from the memory cell should be used to produce the output at the current time step. The gating mechanisms of LSTMs enable them to selectively store or discard information over long periods of time, allowing them to capture long-term dependencies in sequential data.

LSTMs have been shown to be highly effective for a wide range of sequential data processing tasks, such as speech recognition, machine translation, and natural language processing. They have also been used in combination with CNNs for image captioning and video classification. Recent advances in LSTM architecture have led to the development of deep LSTM networks, which have been shown to achieve state-of-the-art performance on a variety of sequential data processing tasks.

Activation function

Activation functions are a fundamental aspect of deep learning models and are integral to the functioning of both convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These functions introduce nonlinearity into the network, allowing it to model and learn complex relationships between input and output data. There are several types of activation functions commonly used in deep learning, including sigmoid, tanh, softmax, and ReLU. Sigmoid 2 and tanh 3 functions map input values to a range between -1 and 1, and 0 and 1, respectively.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3}$$

The softmax function 4 is used in the output layer of classification tasks to convert the output of the previous layer into probabilities that sum to 1.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{4}$$

ReLU 1 is widely used in CNNs due to its simplicity and computational efficiency, and it maps input values below zero to zero, while retaining positive values. The selection of an activation function depends on the specific application and the properties of the data being analyzed. Overall, activation functions play a critical role in deep learning models, enabling them to capture complex patterns and relationships in data, and therefore, they are essential in the engineering of robust and efficient deep learning models.

Optimization Algorithm

Optimization algorithms play a crucial role in training the model effectively. The optimization algorithm is responsible for adjusting the parameters of the model (i.e., weights and biases) in such a way that the loss function is minimized. There are various optimization algorithms available, and selecting the appropriate one is essential for achieving good performance.

One of the most commonly used optimization algorithms is stochastic gradient descent (SGD), which updates the weights of the network based on the gradient of the loss function with respect to the weights. However, SGD has some limitations, such as being sensitive to the learning rate and often getting stuck in local minima. Therefore, variants of SGD have been developed, such as Adam and RMSProp, to improve its performance.

Adam (Adaptive Moment Estimation) is an optimization algorithm that adapts the learning rate for each weight based on the first and second moments of the gradients. This allows Adam to adjust the learning rate on a per-parameter basis, making it more effective than traditional SGD in many cases. Here is the algorithm for Adam optimization algorithm since it is the main one we will be using in our research:

- Calculate the gradient of the loss function with respect to the weights:
  $g = \nabla L(w)$
- Update the biased first moment estimate:
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g$
- Update the biased second raw moment estimate:
  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g^2$
- Compute the bias-corrected first and second moment estimates:
  $m_{\text{hat},t} = \frac{m_t}{1-\beta_1^t} v_{\text{hat},t} = \frac{v_t}{1-\beta_2^t}$
- Update the weights:
  $w_{t+1} = w_t - \alpha \frac{m_{\text{hat},t}}{\sqrt{v_{\text{hat},t}} + \epsilon}$

where:

- $\nabla L(w)$ is the gradient of the loss function L with respect to the weights w.
- β_1 and β_2 are exponential decay rates for the first and second moment estimates, respectively.

- m_t and v_t are the biased first and second moment estimates at time step t.
- $m_{\text{hat},t}$ and $v_{\text{hat},t}$ are the bias-corrected first and second moment estimates.
- α is the learning rate.
- ϵ is a small constant added for numerical stability.

RMSProp (Root Mean Square Propagation) is another optimization algorithm that is designed to overcome some of the limitations of traditional SGD. RMSProp uses a moving average of the squared gradient to normalize the learning rate, which helps prevent the learning rate from becoming too large or too small.

Choosing the appropriate optimization algorithm for a model is crucial for achieving good performance. The choice of optimization algorithm can significantly impact the convergence rate, stability, and final performance of the model. Therefore, it is essential to carefully consider the properties of different optimization algorithms and their suitability for a specific problem.

Regularization Techniques

Regularization is a critical aspect of training deep neural networks such as CNNs, RNNs, LSTM, and transformer models. It helps to reduce overfitting by introducing constraints on the network's parameters during the training process. There are several regularization techniques that are commonly used in deep learning, including

- Dropout: Dropout randomly drops out a fraction of the neurons during training, forcing the network to learn more robust representations that generalize well to unseen data.
- Early stopping: This technique involves monitoring the performance of the network on a validation set during training and stopping the training process when the validation error starts to increase, indicating that the model is overfitting.
- Data augmentation: Data augmentation involves applying random transformations to the input data during training, such as flipping, rotating, or scaling the images. This helps to increase the diversity of the training data and improve the network's ability to generalize to unseen data.

In summary, regularization techniques play a critical role in preventing overfitting and improving the generalization performance of deep neural networks, including CNNs, RNNs, LSTM, and transformer models. A combination of these techniques was considered during the implementation of our model and were necessary for achieving good results.

Advantages and Limitations of Deep Learning

As we mentioned earlier, deep learning has become a popular research area in recent years due to its impressive performance in a variety of complex tasks. Understanding these advantages and limitations is crucial for to effectively develop and apply deep learning models to solve real-world problems. Therefore, it is important to carefully consider these factors when deciding on the most appropriate deep learning approach for a given task.

Advantages:

- Improved accuracy: Deep learning models have shown exceptional accuracy in many complex tasks such as image and speech recognition, machine translation, and natural language understanding.
- Robustness: Deep learning models can handle noisy, incomplete, and complex data more effectively than traditional machine learning algorithms, making them more robust in real-world scenarios.
- Feature extraction: Deep learning models are capable of automatically learning relevant features from raw data, reducing the need for manual feature engineering.
- Scalability: Deep learning models can scale to large datasets and high-dimensional data with many parameters, which makes them suitable for big data applications.
- Transfer learning: Pre-trained deep learning models can be fine-tuned for new tasks, which reduces the need for large labeled datasets and can speed up the development of new applications.

Limitations:

- Data requirements: Deep learning models require a large amount of labeled data for training, which can be time-consuming and expensive to acquire.
- Computational complexity: Deep learning models are computationally intensive and require specialized hardware and software to train and deploy, which can be costly.
- Interpretability: Deep learning models are often described as "black boxes" because their internal workings are difficult to understand and interpret, making it hard to understand how they arrive at their decisions.
- Overfitting: Deep learning models can easily overfit to the training data, which can result in poor generalization performance on new data.
- Vulnerability to adversarial attacks: Deep learning models are vulnerable to adversarial attacks, where small perturbations to the input can cause the model to misclassify the input.

In summary, deep learning has many advantages over traditional machine learning approaches, but it also has some limitations that need to be considered when applying it to real-world problems. Addressing these limitations is an ongoing area of research, and as deep learning continues to evolve, it is likely that many of these limitations will be addressed.

2.3 Related topics

In this section, we will provide an in-depth overview of several important background topics that are essential to understanding the details of our research. We will cover a range of topics related to Convolutional Neural Networks (CNNs), including the

backpropagation algorithm used to train them, as well as other related topics such as Long Short-Term Memory (LSTM) networks, Transformer models, and various performance metrics. Our goal is to provide a comprehensive understanding of these concepts to ensure that you can fully comprehend the technical aspects of our research.

2.3.1 Convolutional Neural Networks

We have earlier explained how Convolutional Neural Networks (CNNs) were revolutionary and achieved state-of-the-art performance on a variety of tasks; in this section we will explain the architecture of CNNs in detail. CNNs are inspired by the structure and function of the visual cortex in the brain, which processes visual information through a hierarchical arrangement of cells with receptive fields of increasing complexity. Similarly, CNNs consist of multiple layers of neurons that learn to extract increasingly abstract and meaningful features from the input images, using a set of learnable filters (also called kernels or weights) that perform convolutions on the input data. In other words, CNNs are designed, at their core, to mimic the architecture of the human visual system.

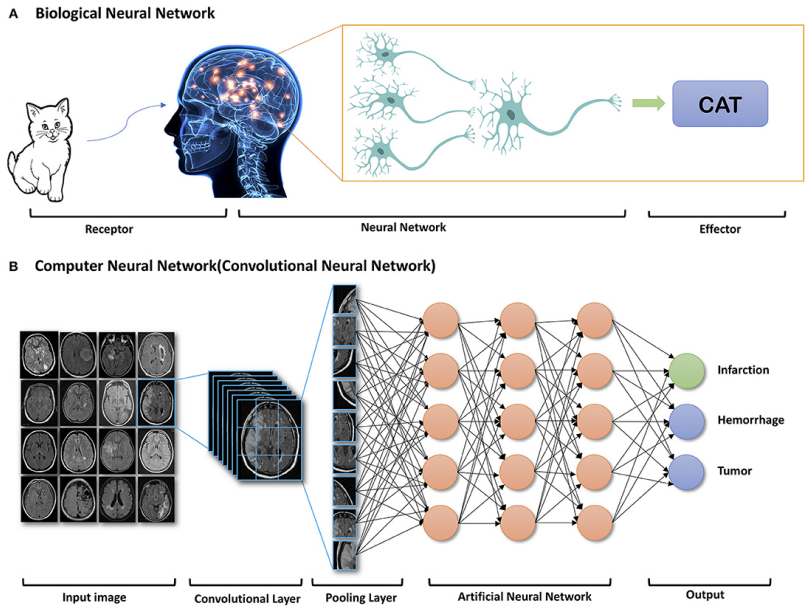

Figure 9: CNNs and brain connecion

Layers

In the context of convolutional neural networks, the first layer is of paramount importance for feature extraction. The set of filters applied to the input image are learned through a training process that optimizes the weights of the filters to identify important features such as edges, or corners in the input image. The output of the convolutional layer is then passed through a nonlinear activation function, such as the Rectified Linear Unit (ReLU), which adds nonlinearity to the network and enables it to learn more complex features.

It's worth noting that the choice of filter size, number of filters, and stride (the distance between successive filter applications) is critical to the performance of the network. Smaller filter sizes allow the network to capture more local features, while larger filter sizes can capture more global features. The number of filters determines the complexity of the learned features, and the stride determines the size of the feature maps produced by the convolutional layer. Please note the structure of the filters in figure 10.

Furthermore, the convolutional layer can also be configured to include padding, which adds additional zeros around the borders of the input image to preserve its spatial dimensions. This can be useful when the input image is too small to produce meaningful feature maps without losing too much spatial information.

Overall, the first layer of a CNN plays a crucial role in feature extraction, and its design requires careful consideration to ensure optimal performance.

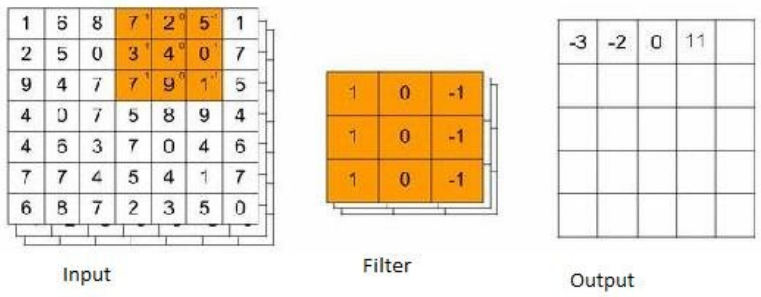

Figure 10: Convolution operation.

Once the low-level features have been identified in the input image, the CNN's subsequent layers are designed to combine them into higher-level features. This is achieved through a process known as pooling, which involves reducing the spatial dimensions of the feature maps by taking the maximum or average value of small regions within them. The purpose of pooling is to make the CNN more robust to small variations in the input image, such as changes in lighting, scale, and orientation.

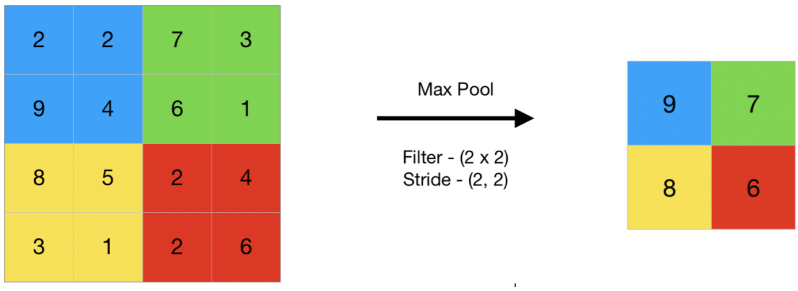

Figure 11: Maxpooling operation.

The combination of convolutional layers and pooling layers allows a CNN to learn increasingly complex features as it progresses through its layers. The final layer of the CNN, shown in figure 12 is typically a fully connected layer, which takes the output

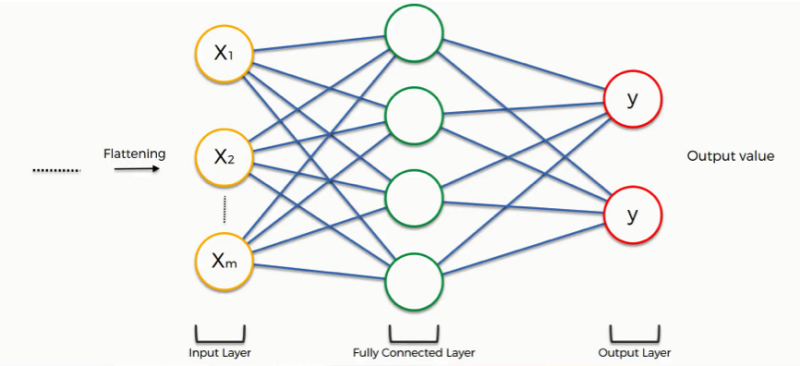

Figure 12: Fully connected layer operation.

of the previous layers and uses it to make a prediction about the input image. The weights of the CNN are learned through backpropagation.

In addition to convolutional and pooling layers, CNNs can also incorporate other types of layers, such as dropout and normalization layers. Dropout layers are used to prevent overfitting by randomly dropping out some of the neurons in the network during training. Normalization layers, such as batch normalization, are used to improve the stability and convergence of the network by normalizing the activations of each layer.

Training process

Now that we have a better understanding of different layers of CNNs, lets discuss the training process, which typically involves two stages: forward propagation and backpropagation. During forward propagation, the input image is passed through the network, and each layer computes its output based on the input from the previous layer. The output of the final layer is then compared to the ground truth label, and the error between them is computed. This error is then used to update the parameters of the network using a technique known as backpropagation.

Backpropagation is the process of computing the gradient of the loss function with respect to the weights of the network, which can be used to update the weights using an optimization algorithm. This is done using the chain rule of calculus. The chain rule allows the gradient to be computed layer-by-layer, starting from the output layer and working backward through the network. This process is often referred to as "backward propagation of errors" because it involves propagating the error in the output layer back through the network to update the weights.

The algorithm then adjusts the weights in the network to minimize the error, and this process is repeated iteratively until the network reaches a satisfactory level of accuracy. Backpropagation is a powerful algorithm that allows CNNs to learn complex features in an image dataset, but it requires a large amount of training data and computational resources.

The backpropagation algorithm has two important components that we need to understand: the loss function and the optimization algorithm.

Loss function

In convolutional neural networks (CNNs), the loss function plays a critical role in determining how well the network performs on a given task. The loss function is used to measure the discrepancy between the predicted output of the network and the actual or ground truth labels. The choice of loss function will depend on the type of problem that the CNN is being trained to solve. For example, if the CNN is being used for classification, then cross-entropy loss is a common choice. Cross-entropy loss 5 is a measure of the difference between the predicted class probabilities and the true class probabilities.

$$\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{5}$$

On the other hand, if the CNN is being used for regression, then mean squared error (MSE) 6 loss may be used. MSE loss measures the average of the squared differences between the predicted and actual values. This is a common loss function used in regression tasks where the goal is to predict a continuous value, such as the price of a house or the temperature at a particular time. The equation for MSE loss is

$$\sum_{i=1}^{D} (x_i - y_i)^2 \tag{6}$$

It is worth noting that there are other types of loss functions that can be used, depending on the specific task at hand. For example, if the CNN is being used for image segmentation, then dice loss may be used. Dice loss measures the overlap between the predicted segmentation and the true segmentation. The choice of loss function in CNNs is crucial to the performance of the network on a given task. Different loss functions are appropriate for different types of tasks, and it is important to select the appropriate loss function to achieve the best performance.

In conclusion, Convolutional Neural Networks (CNNs) have been shown to be revolutionary and achieve state-of-the-art performance on a variety of tasks. The architecture of CNNs is inspired by the hierarchical structure and function of the visual cortex in the brain, with multiple layers of neurons that extract increasingly abstract features from input images. The first layer of a CNN is crucial for feature extraction, and its design requires careful consideration for optimal performance. The subsequent layers of a CNN use a combination of convolutional and pooling layers to learn increasingly complex features, with the final layer being a fully connected layer that makes a prediction about the input image. The training process of a CNN involves forward and backpropagation, with backpropagation updating the weights of the network using the gradient of the loss function. CNNs continue to be an active area of research in the field of deep learning, with many new architectures and applications being developed.

2.3.2 LSTM Network

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) architecture that is particularly well-suited to tasks that require processing and prediction of sequential data. The architecture of an LSTM network consists of multiple LSTM cells that can store information over time and selectively forget or update this information based on input signals. The ability of LSTM networks to

capture long-term dependencies in sequences makes them suitable for applications such as speech recognition, language modeling, and machine translation.

LSTM networks were first introduced in 1997 by Hochreiter and Schmidhuber as an extension of the standard RNN architecture 2.2. The main limitation of standard RNNs is that they suffer from the vanishing gradient problem 13, which occurs when the gradients propagated through the network become so small that they effectively vanish as they travel back in time. This can make it difficult for the network to learn long-term dependencies in sequences. LSTM networks 14 were designed to overcome

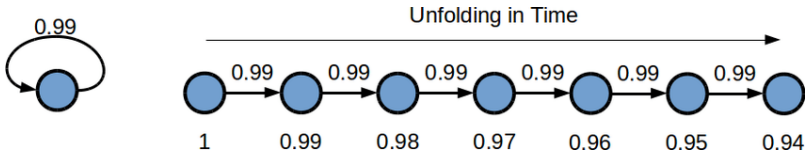

Figure 13: Vanishing gradient problem illustration

this problem by incorporating a memory cell that can selectively store or discard information over time. The basic building block of an LSTM network is the LSTM cell, which consists of three gates (input, forget, and output) and a memory cell that stores information over time. The input gate controls the flow of information into

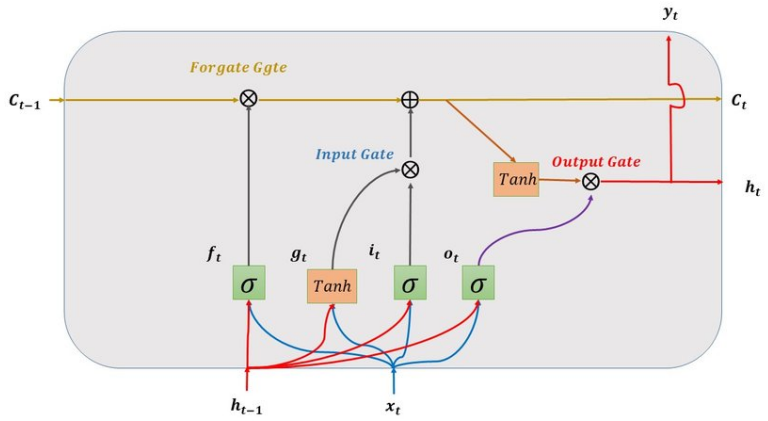

Figure 14: LSTM Network

the memory cell. It takes as input the current input vector and the previous hidden state of the cell, and applies a sigmoid activation function 2 to generate a gate value between 0 and 1 for each element in the input vector. These gate values determine which elements of the input vector are allowed to pass through the gate and update the memory cell.

The forget gate controls the flow of information out of the memory cell. It takes as input the current input vector and the previous hidden state of the cell, and applies a sigmoid activation function to generate a gate value between 0 and 1 for each element in the memory cell. These gate values determine which elements of the memory cell should be forgotten or retained.

The output gate controls the flow of information from the memory cell to the output. It takes as input the current input vector and the previous hidden state of

the cell, and applies a sigmoid activation function to generate a gate value between 0 and 1 for each element in the memory cell. These gate values determine which elements of the memory cell should be passed through the output gate and contribute to the output of the LSTM cell.

The memory cell stores information over time and is updated by the input and forget gates. The update to the memory cell is determined by a candidate vector that is generated by applying a hyperbolic tangent activation function to a linear combination of the current input vector and the previous hidden state of the cell. The input gate then determines which elements of the candidate vector are allowed to update the memory cell.

The output of an LSTM cell is the current hidden state, which is a combination of the updated memory cell and the output gate. The hidden state is then passed on to the next LSTM cell in the sequence, or to the output layer if it is the final cell.

LSTM networks can be stacked to create deep LSTM architectures that can learn hierarchical representations of sequential data. In a stacked LSTM network, the output of one LSTM layer, which was generated using the contribution of the output gate, is fed as input to the next LSTM layer in the stack. This allows the network to learn increasingly abstract representations of the input sequence as it passes through the layers.

LSTM networks could also be Bidirectional, which is an LSTM type that can process the input sequence in both forward and backward directions, allowing them to capture dependencies that span both past and future contexts. Compared to unidirectional LSTM networks, which can only process the input sequence in one direction, bidirectional LSTMs can take into account both past and future contexts simultaneously. As a result, they are better suited for sequence prediction tasks that require context from both past and future time steps. Bidirectional LSTMs are known to improve the accuracy of sequence prediction tasks relative to unidirectional LSTMs, and are particularly useful when modeling complex relationships between input and output data.

Like CNNs, the training of LSTM networks involves optimizing a loss function that measures the difference between the predicted output and the ground truth output. The loss function is typically minimized using backpropagation through time (BPTT).

Backpropagation through time (BPTT) is a commonly used algorithm for training recurrent neural networks (RNNs) in general. It is a variant of backpropagation that allows the network to learn from sequential data, such as time series or natural language.

The basic idea behind BPTT is to unfold the RNN over time, creating a chain of interconnected copies of the network that can be trained using standard backpropagation. Each copy of the network is responsible for processing one time step of the input sequence.

In figure 15, x_i represents the input at time step i, h_i represents the hidden state of the RNN at time step i, and L_i represents the output at time step i. The arrows represent the connections between the layers of the RNN. During the forward pass of BPTT, the RNN processes each input in sequence, updating its hidden state and output at each time step. The output L_i is compared to the target output y_i, and the error is calculated.

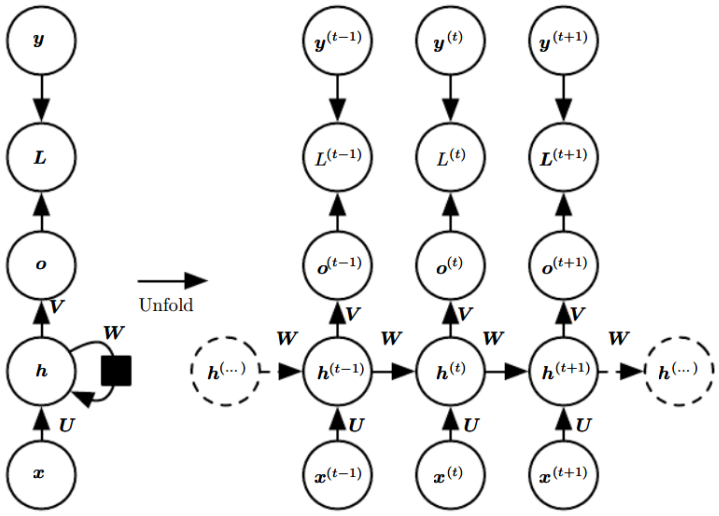

Figure 15: BPTT

During the backward pass of BPTT, the error signal is propagated backwards through time, from the last time step to the first. The weights of the RNN are updated using the chain rule of calculus, similar to standard backpropagation.

By unfolding the RNN over time, BPTT allows the network to learn from sequential data and capture dependencies between past and future inputs.

One challenge with training LSTM networks is the potential for overfitting, which occurs when the network becomes too complex and starts to memorize the training data rather than generalizing to new data. To address this, various regularization techniques can be used, such as dropout, and early stopping discussed earlier.

In addition to these regularization techniques, and similarly to CNNs, LSTM networks can also benefit from various optimization algorithms that are designed to speed up the training process and improve the convergence of the network weights. Popular optimization algorithms for LSTM networks include stochastic gradient descent (SGD), Adam, and RMSprop explained earlier.

Overall, LSTM networks are a powerful tool for processing and predicting sequential data. Their ability to selectively store and forget information over time makes them well-suited to tasks that involve long-term dependencies, such as speech recognition, language modeling, and machine translation. With the right architecture, regularization techniques, and optimization algorithms, LSTM networks can achieve state-of-the-art performance on a wide range of sequential data tasks.

2.3.3 CNNLSTM network

As mentioned earlier, Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks are two powerful deep learning architectures that have shown great success in various applications, from image recognition to natural language processing. CNNs are particularly useful for processing data with a grid-like topology, such as images, while LSTMs are designed to handle sequential data, such as speech or text. However, in some cases, the data may have both spatial and temporal dimensions, and in such cases, a combination of CNNs and LSTMs, called

CNNLSTM, can be used to effectively model and analyze the data.

CNNLSTM is a hybrid neural network architecture that combines the strengths of CNNs and LSTMs to process spatiotemporal data. It is commonly used in applications such as action recognition, video analysis, and gesture recognition, where the data is represented as a sequence of images or video frames. The main idea behind CNNLSTM is to use a CNN to extract features from each image frame and then feed these features into an LSTM to model the temporal dynamics of the sequence. The

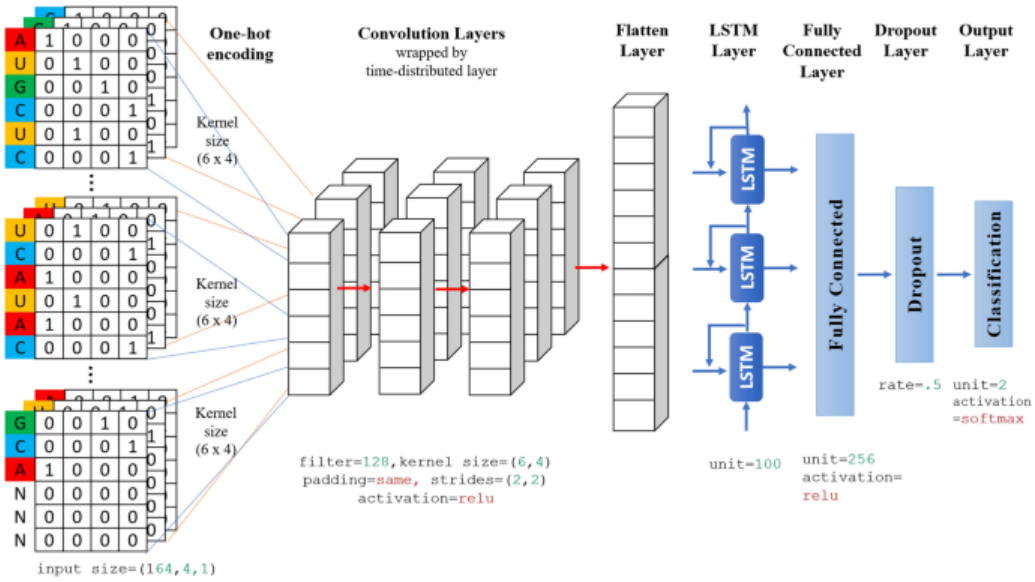

Figure 16: CNNLSTM architecture

first step in building a CNNLSTM network is to define the CNN component responsible for extracting relevant features from each image frame in the sequence. This is done by applying a series of convolutional filters to the input image that produces activation maps that are then passed through a pooling layer. The output of the pooling layer is then flattened and fed into a fully connected layer, which produces a set of feature vectors that represent the image frames. Please refer to 2.3.1 for more details on how that is done.

Once the CNN has extracted the relevant features from each image frame, the next step is to feed these features into an LSTM network. The LSTM is responsible for modeling the temporal dynamics of the sequence and capturing the long-term dependencies between the frames. This is done by maintaining a memory cell that stores information from previous time steps and using a set of gates to control the flow of information into and out of the cell. Please refer to 2.3.2 for more details on how LSTM work.

The CNNLSTM architecture combines the CNN and LSTM components by feeding the feature vectors produced by the CNN into the LSTM at each time step. The LSTM then updates its memory cell and hidden state based on the current input and the previous state, and outputs a prediction for the current time step. The prediction can be a classification label, a regression value, or any other output that is appropriate for the specific application.

Advantages

One of the main advantages of the CNNLSTM architecture is that it can effectively capture both spatial and temporal information in the data. The CNN component is able to extract spatial features from the image frames, while the LSTM component is able to model the temporal dynamics of the sequence. This allows the network to learn complex spatiotemporal patterns that may be difficult to capture with either CNNs or LSTMs alone.

Another advantage of CNNLSTM is that it can handle variable-length sequences. In many applications, the length of the input sequence may vary, and traditional neural network architectures may not be able to handle this variability. However, the LSTM component of CNNLSTM is specifically designed to handle sequential data of variable length, making it well-suited for these types. For these reasons, CNNLSTMs seemed like a good candidate for our chosen model. We will talk more in the following sections about the process of selecting the suitable model.

2.3.4 Transformer Models

Transformer models are a type of neural network architecture that was introduced in a paper by Vaswani et al. in 2017. Transformers are widely used for natural language processing (NLP) tasks, such as machine translation and text classification, but they can also be used for computer vision tasks, such as object detection and image classification, which are more relevant to our problem.

The transformer model architecture is based on the concept of self-attention. Self-attention is a mechanism that allows the model to attend to different positions in the input sequence to compute a representation of that sequence. In the transformer model, the self-attention mechanism is used to compute the contextualized representation of each token in the input sequence.

The transformer model 17 consists of an encoder and a decoder. The encoder takes the input sequence and computes a sequence of contextualized representations. The decoder takes the output of the encoder and generates the output sequence. Both the encoder and decoder consist of a stack of identical layers. Each layer has two sub-layers: a multi-head self-attention mechanism and a position-wise feedforward network. The multi-head self-attention mechanism computes the self-attention for each token in the input sequence. The mechanism applies three linear transformations to the input sequence to project the query, key, and value vectors. These vectors are then used to compute the attention weights, which determine the importance of each token to the representation of the current token. The attention weights are used to compute a weighted sum of the value vectors, which gives the contextualized representation of the current token.

The position-wise feedforward network is a two-layer feedforward network that applies the same transformation to each position in the sequence independently. The output of the feedforward network is added to the output of the multi-head self-attention mechanism to form the output of the layer.

The transformer model is trained using the backpropagation algorithm. The loss function is usually cross-entropy, and the optimization algorithm is usually Adam. During training, the model uses teacher forcing, which means that the decoder inputs are the ground truth outputs shifted by one position.

To make the transformer model more efficient, several engineering tricks have

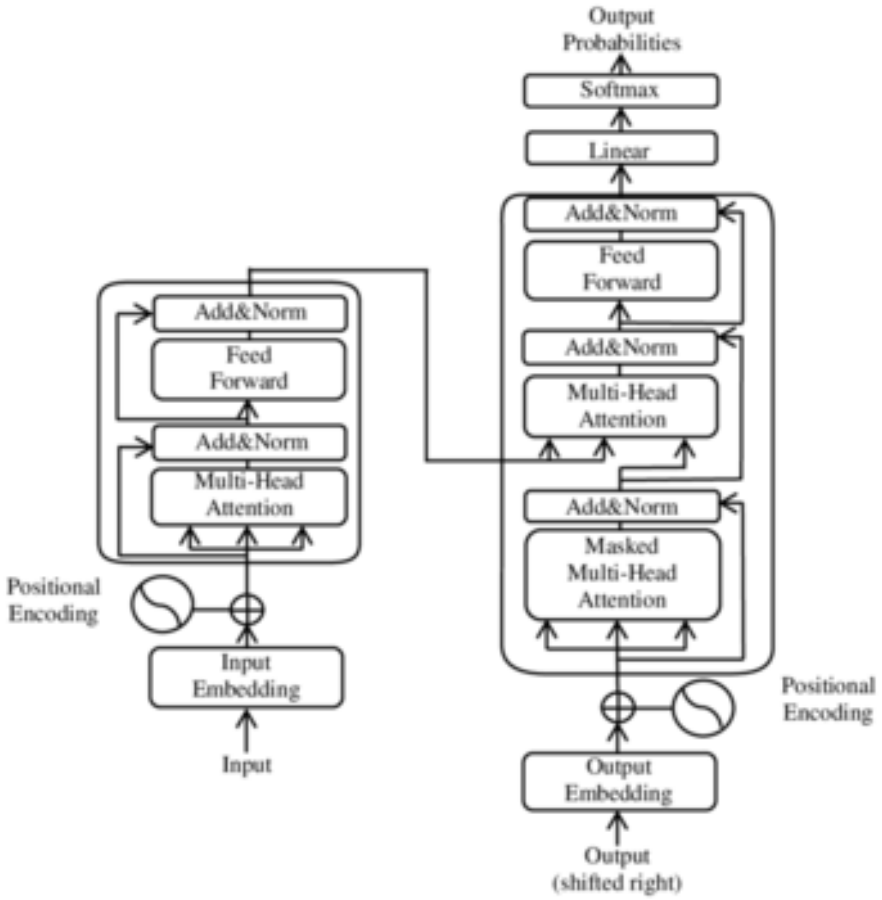

Figure 17: Transformer model architecture

been proposed. One of these tricks is to use a technique called tokenization, which breaks down the input sequence into tokens. Each token represents an image or sub-image in the input sequence. Another trick is to use positional encodings, which encode the position of each token in the sequence. This allows the model to distinguish between tokens that have the same representation but occur at different positions in the sequence.

In conclusion, the transformer model is a powerful neural network architecture that has revolutionized the field of natural language processing. It is based on the concept of self-attention and consists of an encoder and decoder. The transformer model is trained using the backpropagation algorithm, and several engineering tricks have been proposed to make it more efficient.

2.3.5 Performance metrics

In any machine learning project, it is essential to evaluate the performance of the model to determine its effectiveness in solving the task at hand. Performance metrics are used to evaluate the effectiveness of a video recognition model. One commonly used metric is the F1-score, which is the harmonic mean of precision and recall. It is particularly useful when the dataset is imbalanced, as it gives equal weight to both false positives and false negatives. Other common performance metrics

include accuracy, precision, recall, and mean average precision (mAP). Accuracy measures the percentage of correctly classified samples, while precision and recall measure the proportion of true positives to false positives and true positives to false negatives, respectively. mAP is often used in object detection and measures the average precision across multiple thresholds. These performance metrics can provide valuable insights into the strengths and weaknesses of a video recognition model and can be used to optimize its performance. For the sake of our research, we will focus more on F-1 score and confusion matrices.

F-1 score

The F-1 score is a measure of a classifier's accuracy, particularly in situations where the classes are imbalanced. It is the harmonic mean of the precision and recall of the classifier. The F-1 score ranges from 0 to 1, where 1 is the best possible score, indicating perfect precision and recall.

Precision 7 is a measure of how many of the positive predictions made by the classifier are actually correct. It is calculated as the number of true positives divided by the sum of true positives and false positives. In other words, precision measures the proportion of positive predictions that are true.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

Recall 8 is a measure of how many of the true positive cases in the dataset were correctly predicted by the classifier. It is calculated as the number of true positives divided by the sum of true positives and false negatives. In other words, recall measures the proportion of positive cases in the dataset that were correctly identified by the classifier.

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

The F-1 score combines precision and recall by taking their harmonic mean. The harmonic mean is used instead of the arithmetic mean because it gives more weight to smaller values. This means that the F-1 score is more sensitive to imbalanced classes, as it penalizes classifiers that only perform well on the majority class.

The formula for calculating the F-1 score is:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{9}$$

The F-1 score is often used in machine learning and data mining to evaluate the performance of classification models. It is particularly useful in situations where the classes are imbalanced, meaning that one class is much more prevalent than the other. In such situations, a classifier that always predicts the majority class can achieve a high accuracy, but may not be useful in practice.

The F-1 score is a more robust measure of a classifier's performance in such situations, as it takes into account both precision and recall. A high F-1 score indicates that the classifier is performing well on both positive and negative cases, while a low F-1 score indicates that the classifier is either not precise or not sensitive enough.

For our video recognition project, the F1-score is a suitable performance metric to evaluate the model's accuracy. Video recognition involves classifying objects and actions in real-world scenarios, which can be challenging due to variations in lighting, camera angles, and object appearance. The F1-score is a useful metric because it takes into account both precision and recall, making it more robust than accuracy alone. In video recognition, false positives and false negatives can have serious consequences, and the F1-score can help balance these errors. By optimizing the F1-score, the model can achieve a good balance between precision and recall, making it more effective at identifying and tracking objects and actions in videos. However, other metrics such as AUC-ROC and confusion matrices can also provide valuable insights into the model's performance and should be considered in conjunction with the F1-score. Additionally, it is important to be aware of any imbalanced classes or skewed datasets and use appropriate techniques such as resampling or cost-sensitive learning to mitigate these effects.

It is important to note that the F-1 score assumes that the classes are equally important. However, in some applications, one class may be more important than the other. For example, in a fraud detection problem, correctly identifying fraudulent transactions may be more important than correctly identifying non-fraudulent transactions. In such cases, it may be necessary to use a different performance metric that takes into account the relative importance of the classes.

Another limitation of the F-1 score is that it assumes that the classification threshold is fixed. In practice, the classification threshold may need to be adjusted depending on the application. For example, in a cancer diagnosis problem, the classification threshold may need to be set lower to avoid missing any positive cases, even if this leads to more false positives.

Finally, it is worth noting that the F-1 score is not a perfect metric and should be used in conjunction with other performance measures. For example, it may be useful to examine the confusion matrix to get a better understanding of the classifier's performance on different classes.

In conclusion, the F-1 score is a widely used performance metric in classification problems, particularly in situations where the classes are imbalanced. It is a more robust measure of a classifier's performance than accuracy alone, as it takes into account both precision and recall. However, it is important to consider the specific problem and the objectives of the model when choosing a performance metric, as different metrics may be more appropriate depending on the context.

Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for a binary or multi-class classification problem.

In the case of binary classification, the confusion matrix has two dimensions, one for the predicted class and one for the actual class. The four entries of the confusion matrix are as follows:

- True Positives (TP): the model correctly predicted the positive class
- False Positives (FP): the model incorrectly predicted the positive class

- True Negatives (TN): the model correctly predicted the negative class
- False Negatives (FN): the model incorrectly predicted the negative class

The overall accuracy of the model can be calculated by summing the diagonal entries of the confusion matrix and dividing by the total number of samples.

In multi-class classification, the confusion matrix is a square matrix with one row and one column for each class. The entries of the confusion matrix are as follows:

- True Positives (TP): the model correctly predicted samples of the i-th class as belonging to the i-th class
- False Positives (FP): the model incorrectly predicted samples of the j-th class as belonging to the i-th class, where j i
- True Negatives (TN): the model correctly predicted samples of classes other than i as not belonging to the i-th class
- False Negatives (FN): the model incorrectly predicted samples of the i-th class as belonging to classes other than i

The diagonal entries of the confusion matrix represent the number of correctly classified samples for each class, while the off-diagonal entries represent misclassifications. The overall accuracy of the model can be calculated by summing the diagonal entries of the confusion matrix and dividing by the total number of samples. Refer to the figure below to see an example of a confusion matrix over a 3-level classification task. In addition to overall accuracy, the confusion matrix can also be used

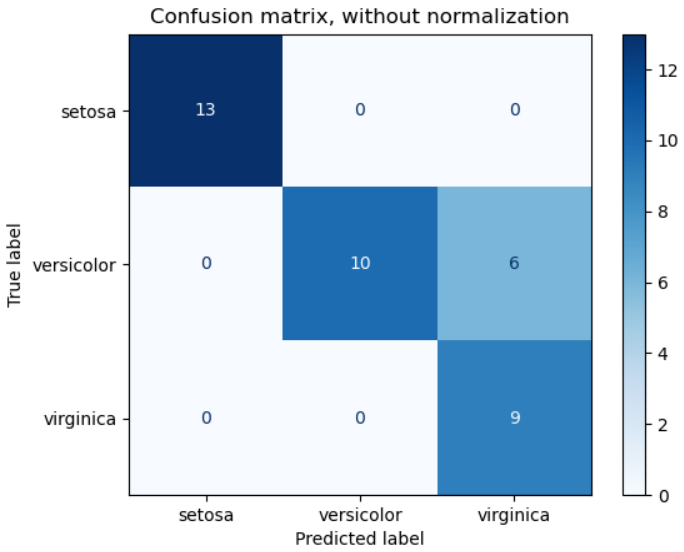

Figure 18: An example of a confusion matrix

to calculate various performance metrics such as precision, recall, and the F1-score for each class. It provides valuable insights into the strengths and weaknesses of a classification model and can be used to identify areas for improvement.

3 Related Work

After having discussed the requisite background information necessary for the successful execution of this research, we will now proceed to examine how other academic works have implemented these concepts in their own investigations, as well as the challenges they encountered in doing so. This section will serve to provide a comprehensive review of relevant literature pertaining to the subject matter, elucidating upon the strategies and methods utilized by other researchers to address similar problems. By synthesizing the findings of previous studies, we hope to establish a foundation for our own investigation and identify gaps in the existing body of knowledge that can be addressed through our research.

3.1 Effects of padding on LSTMs and CNNs

The paper "Effects of padding on LSTMs and CNNs" by Dwarampudi, Mahidhar, and Reddy explores the impact of padding on the performance of LSTMs and CNNs. The authors demonstrate that padding has a significant effect on the performance of these models in terms of accuracy and computational complexity.

In the paper, the authors first provide an introduction to the concept of padding and its importance in deep learning models. Padding involves adding zeros to the input data to ensure that the output of the model has the same size as the input. The authors then describe their experiments, which involve training LSTMs and CNNs on various datasets with and without padding. The datasets used include the IMDB movie review dataset, the MNIST dataset, and the CIFAR-10 dataset.

For the LSTM experiments, the authors train models with varying numbers of hidden layers and varying numbers of units per layer. They find that padding improves the accuracy of the models in all cases, with the greatest improvements seen in models with more hidden layers and units per layer. The authors also note that the use of padding increases the computational complexity of the models, as more zeros are added to the input data.

For the CNN experiments, the authors train models with varying numbers of layers and varying filter sizes. They find that padding also improves the accuracy of the models in all cases, with the greatest improvements seen in models with larger filter sizes. The authors note that the use of padding increases the computational complexity of the models, but that the increase is relatively small compared to that seen in the LSTM experiments.

The authors conclude by discussing the implications of their findings for the design of deep learning models. They note that while padding can improve the accuracy of models, it also increases computational complexity. Therefore, the decision to use padding should be made based on the specific requirements of the application. They also suggest that future research should explore alternative padding methods that can improve the efficiency of models without sacrificing accuracy.

Based on the findings of this paper, it is clear that padding plays an important role in improving the accuracy of LSTMs and CNNs. The use of padding ensures that the output of the model has the same size as the input, which in turn helps the model learn more accurate representations of the data. However, it is important to note that padding also increases the computational complexity of the models. Therefore,

when designing deep learning models, it is important to carefully consider the trade-off between accuracy and computational efficiency when deciding whether or not to use padding.

3.2 Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison

In recent years, there has been a growing interest in developing technology to aid the communication of the deaf and hard of hearing community. Sign language is the primary mode of communication for many of these individuals, and there has been a concerted effort to develop accurate and efficient sign language recognition systems. Pigou et al. (2018) contribute to this effort by presenting a new large-scale dataset and a comparative study of different methods for word-level deep sign language recognition from video.

The dataset, called RWTH-PHOENIX-Weather 2014, consists of over 10,000 video sequences of German sign language gestures, performed by 20 different signers, with annotations for 984 different signs. This dataset is the largest of its kind, and is more diverse and challenging than existing sign language datasets.

The authors then conduct a comparative study of several state-of-the-art deep learning methods for sign language recognition, including 2D and 3D convolutional neural networks (CNNs), and long short-term memory (LSTM) networks. They evaluate the performance of these methods on the RWTH-PHOENIX-Weather 2014 dataset using various evaluation metrics, including recognition accuracy, F1-score, and confusion matrices.

Their results show that 3D CNNs outperform 2D CNNs and LSTMs in terms of recognition accuracy, achieving a mean accuracy of 82.7%. The authors also conduct an ablation study to investigate the impact of different design choices, such as the number of layers and filter sizes, on the performance of the 3D CNNs. They find that deeper and wider networks generally lead to better performance, but there are diminishing returns beyond a certain point.

Despite the promising results, the authors also identify several challenges and limitations of their approach. One of the main challenges is dealing with the large amount of variability in sign language gestures, which can be affected by factors such as the signer's age, gender, and style. They also note that the dataset is heavily imbalanced, with some signs occurring much more frequently than others, which can affect the performance of the models. Additionally, the authors highlight the need for more research on cross-lingual sign language recognition, as their dataset and methods are specific to German sign language.

In conclusion, Pigou et al. (2018) provide a significant contribution to the field of sign language recognition with their new large-scale dataset and comparative study of deep learning methods. Their results demonstrate the effectiveness of 3D CNNs for word-level sign language recognition, while also highlighting the challenges and limitations of their approach. This research provides a valuable foundation for future work in this field, and underscores the importance of developing accurate and efficient sign language recognition systems to improve the communication and quality of life of the deaf and hard of hearing community.

3.3 Real-time continuous sign language recognition from video using spatiotemporal representations

The paper "Real-time continuous sign language recognition from video using spatiotemporal representations" by Donahue et al. (2017) explores the use of spatiotemporal representations for real-time continuous sign language recognition from video. The paper addresses the challenges of capturing the continuous nature of sign language, where signs blend together and the boundaries between them are unclear, as well as the need for real-time processing to enable practical use in applications such as human-computer interfaces and accessibility technology.

The authors used two different datasets: the RWTH-BOSTON-50 dataset, which consists of 50 signs performed by 20 different people, and the RWTH-BOSTON-104 dataset, which includes 104 signs performed by 24 different people. Both datasets were recorded with two cameras to capture different viewpoints, and the signs were performed continuously without pauses between them.

The authors used a 3D convolutional neural network (CNN) to learn spatiotemporal representations from the video data. The CNN took as input sequences of video frames and outputted a sequence of sign labels, one for each frame. The authors experimented with different architectures, including a single-stream network that processed each camera view separately and a two-stream network that combined the two views. They also experimented with different input representations, including raw RGB frames, optical flow, and depth maps.

The authors evaluated their approach using cross-validation, training and testing on different subsets of the data. They achieved state-of-the-art results on both datasets, with an accuracy of 99.3% on the RWTH-BOSTON-50 dataset and 95.2% on the RWTH-BOSTON-104 dataset. The authors also demonstrated real-time performance, achieving a processing speed of 30 frames per second on a GPU. These are really promising accuracies; however, it is worth remembering that the dataset was very small.

One of the challenges the authors faced was the need to handle the continuous nature of sign language. They addressed this by using a sliding window approach to divide the video sequences into shorter segments, and then combining the predictions from adjacent segments to form the final sign labels. Another challenge was the lack of training data, particularly for signs that are less frequently used. The authors addressed this by augmenting the data with synthesized examples, generated by warping and blending existing examples.

In conclusion, the paper "Real-time continuous sign language recognition from video using spatiotemporal representations" by Donahue et al. (2017) presents a novel approach to continuous sign language recognition using spatiotemporal representations and 3D CNNs. The paper addresses the challenges of capturing the continuous nature of sign language and achieving real-time performance, and achieves state-of-the-art results on two different datasets. The authors also address the challenge of training data scarcity by augmenting the data with synthesized examples. This paper provides valuable insights for the development of real-world sign language recognition systems for practical applications.

3.4 Deep Sign: Hybrid CNN-HMM for American Sign Language Classification

Yoo et al. (2017) present a new approach for American Sign Language (ASL) recognition, which they call Deep Sign. The method uses a hybrid convolutional neural network-hidden Markov model (CNN-HMM) architecture to classify hand gestures in real-time video data. The authors claim that this method is faster and more accurate than previous methods, making it suitable for practical applications such as human-computer interaction and communication aids for the deaf.

The authors first collected a dataset of 30 hand gestures, including both isolated and continuous signing, performed by 10 signers. They used an infrared camera to capture the images, which were preprocessed to extract the hand regions and normalize the size and orientation. The dataset was split into training and testing sets, with the testing set consisting of 10% of the data.

The CNN-HMM architecture consists of two main components: a CNN that extracts features from the hand gesture images and an HMM that models the temporal dynamics of the signing sequence. The CNN consists of three convolutional layers followed by two fully connected layers. The HMM consists of three states, corresponding to the beginning, middle, and end of the signing sequence, and each state is modeled by a Gaussian mixture model. The CNN outputs a feature vector for each frame of the video, which is used as input to the HMM.

The authors trained the Deep Sign model using the collected dataset and compared its performance to several other ASL recognition methods, including a rule-based method and a neural network method. The authors evaluated the methods on both isolated and continuous signing tasks and measured their accuracy and speed. They found that the Deep Sign method achieved the highest accuracy on both tasks and was faster than the other methods.

The authors also performed several experiments to evaluate the performance of the individual components of the Deep Sign model. They found that the CNN was able to extract discriminative features from the hand gesture images, and the HMM was able to model the temporal dynamics of the signing sequence. They also found that the number of Gaussian components used in the HMM had a significant impact on the performance of the model.

One of the main problems that the authors faced was the small size of their dataset. They note that their dataset consisted of only 30 hand gestures performed by 10 signers, which may not be representative of the full range of signing variations in ASL. They suggest that a larger and more diverse dataset would be needed to fully evaluate the performance of the Deep Sign method.

In conclusion, Yoo et al. (2017) present a new approach for ASL recognition using a hybrid CNN-HMM architecture. The method achieves high accuracy and real-time performance, making it suitable for practical applications. The authors also identify the need for larger and more diverse datasets to further evaluate the performance of the method.

3.5 Related work contribution to this thesis

Overall, these studies demonstrate the effectiveness of deep learning techniques for word-level ASL recognition. The use of 3D CNNs have proved to be effective; however, the size of their dataset was relatively slower than ours which is the main reason it was feasible. On the other hand, compining 2D CNNs and some sequence recognition networks, like LSTMs. for feature extraction and temporal modeling, respectively, has proven to be a successful approach for this task. Additionally, the development of large-scale datasets has enabled researchers to train and evaluate deep learning models for ASL recognition, leading to significant progress in this field.

4 Methodology and Implementation

To build up on what we have studied so far, we performed 4 main steps: collecting and pre-processing the dataset, preparing the dataset in a format that can be used by a deep learning model, design 2 models architecture and experimenting on their hyper-parameters' values to decide on the best ones, and train and evaluate the constructed models. In this section, we will dive deep into the details of all these steps from the very first process till we got to the experiments that helped us find the results that determined our best performing model.

4.1 Collecting and pre-processing the Dataset

As indicated, the initial phase in our methodology entails gathering a reliable dataset and processing it into a suitable format that can be utilized and fed into the model for training. Our dataset is composed of two distinct components, namely the x-values and the y-values. Specifically, the x-values denote the video inputs, while the y-values represent the corresponding word that each value conveys. By carefully curating and preparing our dataset in this manner, we can enable our model to learn and make accurate predictions based on the relationships between the inputs and outputs.

4.1.1 Loading the Dataset

In the initial stages of our process, we proceeded to collect the dataset, as outlined in the introductory section, which was sourced from a previously published research paper. To begin processing this dataset, we first loaded it into the memory using the available CSV file formats. Next, we utilized the information contained in the CSV file in conjunction with the video recordings provided to extract several key features.

- Sign gloss, which denotes the label or word represented by each snippet of the video, as well as the scene and session in which the sign was performed.
- The scene and session where the sign was performed.
- A sequence of frames from the identified scene, starting from the first frame of the sign gloss and extending to the last frame of the sign gloss.

4.1.2 Editing the shape of the input

To ensure the optimal performance of our model and achieve superior results, we refined the raw input by employing a range of engineering techniques. Specifically, we recognized that the raw input in its original state would be somewhat challenging for our model to process effectively, and as such, we implemented several pre-processing steps to streamline the training process.

One of the key pre-processing steps involved the cropping of the extracted frames into a central square that focused solely on the person performing the sign, while excluding as much of the background as possible. By minimizing the influence of irrelevant background data, we can enhance the accuracy of our model's predictions.

After cropping, the frames were then normalized to ensure consistency in the data representation. To achieve this, we represented each frame as a 3D array, with each element in the array corresponding to a specific pixel in the image. The resulting 3D array representation enabled us to standardize the data across all frames and ensure that our model can effectively learn from the input data. A visual representation of the 3D array representation is shown in Figure 19.

Through these pre-processing steps, we were able to significantly improve the efficiency and effectiveness of our model, ultimately enabling us to make more accurate predictions regarding the sign language used in the provided video snippets.

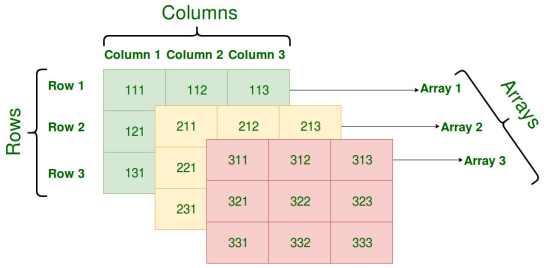

Figure 19: Visualization of a 3D array representation of a frame.

Another utilized technique involved transforming the video frames from their original RGB form into a grayscale, resulting in a single output layer. This transformation was necessary to simplify the data representation, reducing the complexity of the input data and making it easier for the model to learn from it.

After transforming the input data, we then applied a normalization equation to scale the values in the input channel to the range [0, 255], which is the range of values that can be represented by an 8-bit image. This normalization equation was represented by the formula

$$Output\_channel = 255 * (Input\_channel - min)/(max - min)$$

min and max represent the minimum and maximum values in the input channel, respectively.

The purpose of normalization is to standardize the range of values in the input channel, which can help improve the performance of our models by making the input data more consistent and reducing the impact of outliers or extreme values. By normalizing the input data, we can also prevent issues with numerical stability during training, which can occur if the input values are very large or very small.

In addition, by subtracting the minimum value from each pixel and dividing by the range $max - min$, we ensure that the minimum value in the input channel is mapped to 0 and the maximum value is mapped to 255, with all other values scaled proportionally between these two extremes. This standardized range of values further enhances the ability of our model to learn from the input data, resulting in more accurate predictions.

4.1.3 Data Augmentation

In order to further improve the robustness and performance of our model, we decided to apply data augmentation techniques to our dataset prior to training. By augmenting our dataset, we aimed to increase the diversity and quantity of training examples and, consequently, enhance the generalization ability of our model. The techniques we selected were chosen based on their relevance to the problem and their ability to introduce new perspectives to the input data.

The first data augmentation technique we employed was rotation. Specifically, we randomly selected sequences of frames from our dataset and created copies of these sequences. We then rotated the copies by a random angle between -30 and 30 degrees, the choice of these values will be further explained in the experiments and results sections. This technique is particularly useful in our case since it allows our model to better handle variations in the orientation of the signer, which is an important factor in sign language recognition.

Another data augmentation technique we employed was color jittering. Similar to the rotation technique, we randomly selected sequences of frames and created copies of these sequences. However, instead of rotating the copies, we changed their contrast using the Python OpenCV library. This technique was chosen because it has been shown to improve the performance of deep learning models in various computer vision tasks.

It is worth noting that not all data augmentation techniques are applicable to our problem. For example, flipping the input frames would not be realistic since sign language gestures are not typically performed in a mirrored or flipped orientation. Overall, by applying these data augmentation techniques, we were able to increase the size and diversity of our dataset, which ultimately led to improved performance and robustness of our model.

4.1.4 Storing the processed Dataset

After refining the Dataset, we stored it in two NumPy arrays to hold the input sequences and their corresponding labels. To be specific, the x-values were 3D arrays of grayscale video frames, and the labels were stored in the other array. However, we faced an issue where the input sequences varied in length. Although LSTM-based models can handle variable-length inputs, research that we have previously discussed in the background has shown that padding the sequences to a uniform length can enhance model performance and accuracy. Furthermore, we aimed to train our dataset on multiple models and ensure its viability for this purpose. Therefore, we addressed this inconsistency by padding all frame sequences with black screens, represented by 2D arrays of zeros. We extended them to match the length of the longest sequence, thereby ensuring that all inputs have the same length, making them suitable for training the LSTM model.

4.2 Choice of Models

Based on our background research, we have determined that the CNNLSTM and Transformer models are the most suitable options for training our dataset. However, we found that their performances on various video datasets were similar, and we were unable to determine which model was better suited for our specific task. As a result, we decided to build both models and conduct extensive experiments on each to determine which was superior. This section will provide a detailed description of the initial architecture of both models, as well as a thorough explanation of the experiments we conducted to optimize their performance. Throughout our experiments, we ensured that we used the same dataset with identical train, validation, and test split ratios to eliminate external factors that could impact model performance. After completing all of our experiments, we selected the optimal model and hyperparameters and tested it with various train, validation, and test split ratios to identify the best split. In the experimentation section, we will delve further into the experiments we conducted to determine the hyperparameters used and the optimal

4.3 CNNLSTM Model Architecture

We decided to use a CNNLSTM model as our first model, based on the reasons outlined in our background research. The model architecture consists of several layers that were defined using the Keras Sequential API, which is a widely used deep learning framework. In the following section, all the values for the hyperparameters used were determined on experiments that will be further explained.

- The first layer of our chosen CNNLSTM model is a ConvLSTM2D layer, which has been defined using the Keras Sequential API. This layer has three filters and a 3x3 kernel size, and it takes as input a sequence of frames with a shape of (SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 1). Since we have grayscale images, the fourth input is set to 1. The activation function used in this layer is the hyperbolic tangent (tanh), and it returns a sequence of output tensors.
  The purpose of this layer is to perform spatiotemporal feature extraction on

the input frames. To determine the appropriate hyperparameters to use, we conducted several experiments. We selected the number of filters and the kernel size based on their effectiveness in extracting features from similar datasets in previous research. Additionally, we experimented with different activation functions and found that some of them performed better than others in our case. By performing these experiments, we were able to optimize the model's performance and ensure that it is suitable for our dataset.

- The second layer in our CNNLSTM model is a MaxPooling3D layer, which is responsible for temporal pooling. It achieves this by taking the maximum value over a 10x10 spatial area and over the time dimension. The pooling size used is (1,10,10), and 'same' padding is applied to maintain the output size equal to the input size. This layer reduces the spatial resolution of the feature maps while preserving the temporal information. Its primary purpose is to extract the most important features from the previous layer's output and prepare them for the next layer.

- The third layer in our model architecture is a TimeDistributed layer with a Dropout regularization of 0.3. This layer applies the dropout mask to each input element of the sequence independently, helping to prevent overfitting by randomly dropping out elements of the input sequence during training. Specifically, the TimeDistributed wrapper ensures that the same dropout mask is applied to every time step of the sequence, as opposed to applying the dropout mask once to the entire sequence. This helps to maintain temporal correlation between the frames in the sequence by ensuring that the same input features are not always dropped out. The dropout regularization rate of 0.3 was chosen based on experiments to balance the trade-off between reducing overfitting and maintaining sufficient model capacity.

- The next two ConvLSTM2D layers with 6 and 12 filters respectively, continue to extract and downsample spatiotemporal features. The choice of filter size and number was determined through experiments as explained earlier. The first layer uses "channels_first" data format, which is a common format for frameworks such as Theano, while the second layer uses "channels_last" data format, which is a common format for frameworks such as TensorFlow.
  To further explain the choice of data format, "channels_first" data format means that the input tensor's shape will be (batch_size, channels, time, height, width), while "channels_last" data format means that the input tensor's shape will be (batch_size, time, height, width, channels). As mentioned, the choice of data format is dependent on the deep learning framework used and the hardware on which it runs. In our case, we decided to use both data formats to ensure compatibility with different frameworks and hardware.

- To further improve the model's performance, we added two more layers that are similar to the previous layers with some differences in their configuration. The next layer is a MaxPooling3D layer with a pooling size of (1,2,2) and

'same' padding. This layer performs temporal pooling by taking the maximum value over a 2x2 spatial area and over the time dimension. This additional pooling helps in further reducing the spatial resolution of the feature maps while preserving more information in the temporal dimension.
The fifth layer is a TimeDistributed layer with a Dropout regularization of 0.4, which is slightly higher than the previous dropout rate used. This layer helps to prevent overfitting by randomly dropping out elements of the input sequence during training, and as we increase the dropout rate, it becomes more likely that any given element will be dropped out. The use of different pooling sizes and dropout rates in the layers helps in making the model more robust and less prone to overfitting.

- The final two layers of the CNNLSTM model are the Flatten and Dense layers. The Flatten layer takes the output of the last ConvLSTM2D layer and flattens it into a 1D vector, which serves as input for the Dense layer. The output size of the Dense layer is equal to the number of classes. The activation function used in the Dense layer is softmax, which is commonly used in multi-class classification tasks. Softmax outputs a probability distribution over the classes, ensuring that the sum of the probabilities of all the classes is equal to one. This allows us to interpret the output of the model as the probability of the input belonging to each of the classes. By choosing softmax as our activation function, we can obtain a clear and interpretable output from the model.

The following pseudo code shows the function that we used to construct our CNNLSTM2D model:

Algorithm 1 CNNLSTM Model

function CNNLSTM MODEL(*output*)
$CNNLSTM \leftarrow Sequential()$
$CNNLSTM \leftarrow CNNLSTM + ConvLSTM2D$
$CNNLSTM \leftarrow CNNLSTM + MaxPooling3D$
$CNNLSTM \leftarrow CNNLSTM + TimeDistributed(Dropout(0.3))$
$CNNLSTM \leftarrow CNNLSTM + ConvLSTM2D$
$CNNLSTM \leftarrow CNNLSTM + MaxPooling3D$
$CNNLSTM \leftarrow CNNLSTM + TimeDistributed(Dropout(0.3))$
$CNNLSTM \leftarrow CNNLSTM + ConvLSTM2D$
$CNNLSTM \leftarrow CNNLSTM + MaxPooling3D$
$CNNLSTM \leftarrow CNNLSTM + TimeDistributed(Dropout(0.3))$
$CNNLSTM \leftarrow CNNLSTM + Flatten()$
$CNNLSTM \leftarrow CNNLSTM + Dense(512)$
$CNNLSTM \leftarrow CNNLSTM + Dense(output\_size)$
return CNNLSTM
end function

4.4 Transformer model

The second model architecture utilized in this study is a ViViT (Vision Vision Transformer) model, which, as explained earlier, is a cutting-edge approach for image classification tasks. This architecture combines convolutional neural networks (CNNs) and transformers to process input frame sequences. The CNNs handle the spatial features of the frames, while the transformers are responsible for capturing the temporal relationships between the frames.
The transformer network of the ViViT model consists of a series of transformer blocks, each of which contains multiple self-attention layers and feedforward layers. The self-attention layers capture the temporal relationships between the input frames, while the feedforward layers process the output of the self-attention layers to generate the final predictions. The output of the final transformer block is passed through a global average pooling layer to obtain a single feature vector, which is then passed through a fully connected layer with softmax activation to obtain the final class probabilities.
The implementation of the ViViT model was done using the PyTorch library, which provides an efficient and flexible framework for deep learning tasks. The code for the ViViT model architecture was adapted from a public GitHub repository, which was developed and maintained by the original authors of the ViViT model. The code was then fine-tuned on the specific dataset used in this study, with hyperparameters and other settings optimized through extensive experimentation. In this section, we will delve into the details of the algorithm utilized for each layer of the ViViT model, and provide a comprehensive explanation of how each layer operates.

- The first layer is a custom layer for performing tubelet embedding on a 3D input tensor representing the video input. Tubelet embedding first divides the video into small spatio-temporal segments (tubelets), and then projecting each tubelet onto a low-dimensional embedding space.
  The TubeletEmbedding layer takes three arguments: embed_dim, which specifies the dimensionality of the embedding space; patch_size, which specifies the size of each tubelet (in space and time); and **kwargs, which allows for any additional keyword arguments to be passed to the layer. The values of these parameter were also chosen based on different experiments.
  In the __init__ method, we initialize two sub-layers: a 3D convolutional layer (layers.Conv3D) for projecting the tubelets onto the embedding space, and a reshape layer (layers.Reshape) for flattening the projected tubelets into a 2D tensor. The Conv3D layer is configured with filters=embed_dim to output embeddings of size embed_dim, and with kernel_size=patch_size, strides=patch_size, and padding="VALID" to ensure that each tubelet is processed independently without overlap.
  In the call method, the layer takes an input tensor videos of shape (batch_size, time_steps, height, width, channels) and first passes it through the Conv3D layer to obtain a tensor of shape (batch_size, num_tubelets, 1, 1, embed_dim), where num_tubelets is the number of tubelets obtained by dividing the input video into non-overlapping segments of size patch_size. The resulting tensor is then flattened using the Reshape layer into a 2D tensor of shape (batch_size, num_tubelets * embed_dim), which represents the tubelet embeddings for the

input video. The flattened tensor is then returned as the output of the layer.

Algorithm 2 Tubelet Embedding

```
function TubeletEmbedding(class)
    function _init_(self, embedDim, patchSize, **kwargs)
        super()._init_(**kwargs)
        self.Projection ← layers.Conv3D(embedDim, patchSize, patchSize)
        self.Flatten ← layers.Reshape((−1, embedDim))
    end function
    function call(self, videos)
        projectedPatches ← self.Projection(videos)
        flattenedPatches ← self.Flatten(projected_patches)
    end function
```

Table 1 shows the experiments done to decide on the most optimal values for this layers hyperparameters.

Parameter	Search Space
embed_dim	32, 64, 128
patch_size	(4,4,4), (8,8,8), (12,12,12)

Table 1: Tublet embedding experiments

- This layer is a PositionalEncoder class that performs positional encoding of the input tokens. The purpose of positional encoding is to provide the model with information about the position of the tokens in the sequence.
  The layer takes as input a sequence of encoded tokens, represented as a tensor of shape (batch_size, num_tokens, embed_dim). The build method of the layer is responsible for creating a position embedding matrix, which is an Embedding layer that maps the position indices to vectors of size embed_dim. The positions are calculated using tf.range and are stored in a tensor of shape (num_tokens,). The call method of the layer applies the position embedding to the input tensor by adding the position embeddings to the encoded tokens. This is done by first computing the position embeddings using the position_embedding layer and then adding them to the encoded tokens. The resulting tensor has the same shape as the input tensor, (batch_size, num_tokens, embed_dim), but with the addition of positional information.

- The main part of the architecture is a series of transformer blocks. Each block applies layer normalization to the input tensor, followed by a multi-head self-attention layer to capture the relationships between patches. The attention output is then added to the input tensor via a skip connection. The resulting tensor is then passed through another layer normalization layer followed by a feedforward neural network. The output of the feedforward network is added to the input tensor via another skip connection. This process is repeated for a specified number of times.

Algorithm 3 Positional Encoder

function PositionalEncoder(*class*)
  function _init_(self, embedDim, **kwargs)
    $super().\_init\_(**kwargs)$
    $self.embedDims \leftarrow embedDim$
  end function
  function build(self, inputShape)
    $--, numTokens, -- \leftarrow inputShape$
    $self.positionEmbedding \leftarrow layer.Embedding(numTokens, self.embedDim)$
    $self.positions \leftarrow tf.range(0, numTokens, 1)$
  end function
  function call(self, encodedTokens)
    $encodedPositions \leftarrow self.positionEmbedding(self.positions)$
    $encodedTokens \leftarrow encodedTokens + encodedPositions$
    return encodedTokens
  end function

After the transformer blocks, the output tensor is passed through another layer normalization layer and a global average pooling layer to produce a single feature vector for each input video. Finally, the feature vector is passed through a dropout layer and a fully connected layer with a softmax activation function to produce a probability distribution over the output classes.

Algorithm 4 Vivit Classifier

function createVivitClassifier(*output*)
  $inputs \leftarrow layers.Input(inputShape)$
  $patches \leftarrow tubeletEmbedder(inputs)$
  $encoded_patches \leftarrow positionalEncoder(patches)$
  for i in range transformerLayers **do**
    $x_1 \leftarrow LayerNormalization(1e-6)(encodedPatches)$
    $attentionOutput \leftarrow MultiHeadAttention(numHeads, embedDim/numHeads)(x_1, x_1)$
    $x_2 = LayerNormalization(1e-6)(x_1)$
    $x_2 = Sequential(Dense, Dense)(x_2)$
    $encodedPatches = layers.Add()(x_2)$
  end for
  $representation \leftarrow LayerNormalization()(encodedPatches)$
  $representation \leftarrow GolbalAvgPool1D()(representation)$
  $representation \leftarrow Dropout(0.3)(representation)$
  $output \leftarrow Dense(numClasses)(representation)$
  $model \leftarrow Model(inputs, output)$
  return model

Overall, this architecture uses a combination of convolutional and transformer layers to capture spatial and temporal information from input videos and make predictions about their content.

4.5 Experiments

In this section, we will provide a detailed description of the experimentation performed to achieve the best results in our implementation. As mentioned earlier, our approach was based on extensive experimentation to determine the best techniques for each phase of the research.

To begin with, we experimented with various pre-processing techniques for our dataset. We tested different image transformations and data augmentation methods to improve the quality and diversity of our dataset. We analyzed the performance of techniques such as cropping, and normalization on our dataset. We also experimented with techniques like random rotation, as well as adjusting brightness and contrast levels.

We also conducted experiments to find the best architecture for our model. We compared the performance of transformer models and CNNLSTM models, which have both shown promising results in similar tasks. We tested various configurations of both models, including different numbers of layers, filters, and kernel sizes. We also experimented with different activation functions, regularization techniques, and learning rates.

To ensure the accuracy and reliability of our results, we followed a general rule of training each variation of our model 5 times and taking the average accuracy as the real accuracy. This allowed us to mitigate any variations or randomness in our experiments and obtain more reliable results.

In summary, our experimentation process was a crucial aspect of our research, as it allowed us to determine the most effective pre-processing techniques and model architectures for our task. The results of our experiments served as the basis for the final implementation of our model.

4.5.1 Pre-processing Experiments

Data pre-processing is a crucial step in achieving optimal performance of our model. Therefore, we devoted significant effort and time to ensure that our dataset is well-suited to our model and provides the best representation of the words that our model is trying to learn.

Augmentation

As previously mentioned, we employed various data augmentation techniques to enhance the robustness of our model. However, not all data augmentation methods are equally effective or relevant for our specific task. For instance, some methods may not be intuitive or may not provide any discernible benefits for our task, like flipping the frames.

To ensure that our data augmentation approach is effective, we incorporated random video rotation with three different ranges of values that are possible in real life scenarios. This allowed us to evaluate the optimal range of rotation angles that should be used for our task. Through experimentation, we found that certain ranges of rotation angles were more effective than others in improving the performance of our model. Our approach is informed by prior research that has shown the effectiveness of data augmentation techniques in enhancing the performance of machine learning

models, particularly in image and video classification tasks. Table 2 shows the experiments done for this part.

Range beginning	Range ending
-10	10
-20	20
-30	30

Table 2: Video rotation variations.

Padding

We earlier discussed that padding our dataset was a crucial step to make it compatible with different models and to enhance the performance of our algorithm. However, the question of whether to use post-padding or pre-padding is not always straightforward and depends on the nature of the task the model aims to solve. In light of this, we conducted a series of experiments to compare the effectiveness of both padding techniques. Specifically, we applied both post-padding and pre-padding to our dataset and trained our models on each variant. We then compared the accuracies of the models

4.5.2 CNNLSTM2D Model Experiments

After conducting experiments on our data pre-processing techniques and obtaining satisfactory results from our initial models, we proceeded to further improve our models' performance by experimenting with their respective layers. In this section, we will detail the experiments performed on the layers of our first model, the Convolutional Neural Network Long Short-Term Memory 2D (CNNLSTM2D).
Our objective was to determine the optimal number of layers and their corresponding parameters for our CNNLSTM2D model. We experimented with varying numbers of layers, kernel sizes, and pooling strategies to find the optimal configuration that would result in the best performance. To achieve this, we used a grid search method to test various combinations of layer parameters.
The selection of optimal layers and their parameters is crucial to the performance of a model. A well-designed model with appropriately chosen layers can reduce overfitting and improve the accuracy of predictions. Through our experiments, we aimed to identify the most effective layer configuration for our specific task.
In the following sections, we will discuss the experiments performed on the layers of our CNNLSTM2D model in detail.

First Layer

In this model, each layer is characterized by a set of hyper-parameters that can significantly impact the performance of the model. Slight variations in these hyper-parameters can either improve or negatively affect the performance of the model. To ensure that our model achieves the best possible performance, we experimented

with different variations of hyper-parameters for each layer of our model. Specifically, we focused on determining the most suitable hyper-parameters for our training problem, using our initial model, CNNLSTM2D, as the basis for our experiments. The different variations of hyper-parameters that we explored in the first layer are summarized in Table 3.

Parameter	Search Space
filters	[1,12]
kernel size	[3x3,6x6]
activation function	tanh, Relu

Table 3: ConvLSTM2D layer parameters

Second Layer

In the MaxPooling3D layer, the primary hyper-parameter of concern is the dimensions of the spatial area. To determine the optimal values of these hyper-parameters, we conducted a series of experiments and varied the values of the width and height dimensions within specific ranges. Our experiments are summarized in Table 4. By analyzing the results of these experiments, we were able to identify the optimal range of dimensions that should be used in our MaxPooling3D layer to achieve the best performance.

Dimension	Range beginning	Range ending
Width	5	10
Height	5	10

Table 4: MaxPooling3D experiments

Third Layer

We conducted experiments on the TimeDistributed layer to investigate the effect of the dropout regularization rate hyper-parameter, a critical hyper-parameter that regulates overfitting in the model, on the performance of our model. To find the optimal value for this hyper-parameter, we experimented with different values ranging from 0.2 to 0.5.

Fourth to Sixth

To ensure consistency and comparability across our models, we performed experiments on these 3 layers in the same manner as the previous ones. This involved varying the hyper-parameters of each layer and evaluating the performance of the model. Specifically, we experimented with the hyper-parameters of the Conv2D layers, and the LSTM layer. The experiments for these layers were conducted in the same way as for the previous layers and the results were recorded for analysis. The values for the hyperparameters used in these layers were determined based on the same experiments explained earlier in tables 3, and 4.

Dense Layer

In a dense layer of a CNNLSTM model, the main hyperparameters that can be experimented with are the number of neurons and the activation function. The number of neurons in a dense layer determines the complexity of the model and its ability to capture more intricate relationships between the input data and the output. The activation function controls the non-linearity of the output, which is crucial in making the model more expressive and able to model complex patterns in the data. Table 5 shows the different variations applied for this layer.

Parameter	Search Space
Activation function	ReLU, Sigmoid, Tanh
No. of Neurons	512, 1024
Number of Dense layer	1, 2, 3

Table 5: Dense layer parameters

4.5.3 Transformer Model Experiments

In the subsequent stage, we conducted experiments on our transformer model. The architecture of this model was designed in such a way that it resulted in a set of hyper-parameters that were defined as global variables. These hyper-parameters were then tuned iteratively until we obtained the optimal performance. Table 6 shows these variations. Note that we end with 4 possible values for 5 different parameters which means a total of 1024 variation which shows the depth of our experiments.

Parameter	Search Space
Transformer_layers	2,4,6,8
Num_heads	4,8,16,32
Embed_dims	64, 128, 256, 512
layer_norm_eps	1e-6, 1e-5, 1e-4, 1e-3
dropout	0.2, 0.3, 0.4, 0.5

Table 6: Transformer Model parameters

The rule of each hyperparameter is explained in the following list:

- transformer_layers: Number of Transformer layers to use.
- num_heads: Number of heads in the Multi-Head Self-Attention layer.
- embed_dim: Dimensionality of the embedding space.
- layer_norm_eps: Epsilon value for layer normalization.
- dropout: Dropout rate for the output layer.

4.6 Results and Analysis

The results section presents the outcomes of our experiments and analyses conducted to evaluate the performance of our models. The primary metric used to assess our models' performance is the F-1 score. In this section, we present the results of our experiments on data augmentation and padding techniques. We also evaluate the performance of our models on the test set and compare them against the baseline models. Finally, we provide an analysis of the results and draw conclusions about the effectiveness of our models in classifying American Sign Language gestures.

4.6.1 Pre-processing Experiments

As mentioned earlier, we experimented with three types of pre-processing: data augmentation, padding, and train, validation, test split ration, and in this section we will show the results of the performed experiments.

Augmentation

The results of our rotation experiments suggest that applying rotation augmentation within the range of -30 to 30 degrees is the most effective for improving the accuracy of both the Transformer and CNNLSTM models. As we can see from Table 7, the accuracy of both models improves as the range of rotation increases. The highest accuracy of 0.79 was achieved by the CNNLSTM model with rotation within the range of -30 to 30 degrees, which is a significant improvement compared to the accuracy of 0.66 with rotation within the range of -10 to 10 degrees.

Range	**Transformer**	**CNNLSTM**
-10 to 10	0.69	0.66
-20 to 20	0.71	0.68
-30 to 30	**0.72**	0.79

Table 7: Results of rotation experiments.

It is worth noting that the Transformer model generally performed worse than the CNNLSTM model in all ranges of rotation, with a maximum accuracy of 0.72. This could be due to the fact that the Transformer model is more sensitive to the distribution of the input data, and rotation augmentation may have disrupted the sequential structure of the data. On the other hand, the CNNLSTM model is better suited for capturing spatiotemporal features and therefore may be more robust to rotation augmentation.

Overall, our rotation experiments demonstrate the importance of data augmentation for improving the accuracy of video recognition models. By rotating the input data within an appropriate range, we can provide the models with more diverse examples to learn from, which can lead to better generalization performance.

Padding

Table 8 shows the results of our padding experiments explained earlier.

The results of our padding experiments (Table 8) show that pre-padding outperforms post-padding for both models. The Transformer model achieved an accuracy

Padding	Transformer	CNNLSTM
Post-padding	0.68	0.65
Pre-padding	**0.74**	0.70

Table 8: Results of padding experiments.

of 0.74 with pre-padding, compared to 0.68 with post-padding, while the CNNLSTM model achieved an accuracy of 0.70 with pre-padding, compared to 0.65 with post-padding.

This suggests that pre-padding may be a more effective technique for enhancing the performance of our models on sign language recognition tasks. This is likely due to the fact that pre-padding provides the model with more information about the start of the sequence, which may be particularly important for sign language recognition tasks where the start of a sign may be a critical cue for accurate recognition.

Overall, the results of our padding experiments suggest that pre-padding may be a useful technique for enhancing the performance of sign language recognition models. Further exploration of different padding techniques and their effects on model performance may lead to even better results.

In summary, our experiments on data augmentation and padding techniques have shown that there are certain configurations that work better for our Transformer and CNNLSTM models. The best range for rotation augmentation was found to be -30 to 30 degrees for both models, with a significant increase in accuracy for the CNNLSTM model. On the other hand, pre-padding was found to be more effective than post-padding for both models, resulting in an accuracy increase of around 6% for the Transformer and 5% for the CNNLSTM.

4.6.2 Model Architecture Experiments

After finding the best hyper-parameters' values for each architecture, and the best methods for pre-processing our dataset, based on the mentioned experiments and results, We experimented with both types of model architectures: transformer and CNNLSTM using the best obtained hyper-parameters' values.

First Layer

In this section, we present the results of our experiments on the impact of different filter sizes, kernel sizes, and activation functions on the performance of the first layer in our CNNLSTM2D.

Filter Size

Table 9 summarizes the results of our experiments on different filter sizes. As shown in the table, the F-1 score increases as we increase the filter size from 1 to 3, then decreases as we increase the filter size from 3 to 5.

Kernel size

Table 10 shows the results of our experiments on different kernel sizes. As shown in the table, the F-1 score is the highest for a 3x3 kernel size.

Filter Size	F-1 Score
1	0.68
3	**0.77**
5	0.71
7	0.71
12	0.68

Table 9: Results of experiments on different filter sizes.

Kernel Size	F-1 Score
3x3	**0.77**
6x6	0.69

Table 10: Results of experiments on different kernel sizes.

Activation Function

Table 11 summarizes the results of our experiments on different activation functions. As shown in the table, the F-1 score is the highest for the tanh activation function.

Activation Function	F-1 Score
ReLU	0.72
tanh	**0.77**
sigmoid	0.68

Table 11: Results of experiments on different activation functions.

The results of our experiments on different filter sizes show that the F-1 score increases as we increase the filter size from 1 to 3, then decreases as we increase the filter size from 3 to 5. For larger filter sizes of 7 and 12, the F-1 score drops further to 0.71 and 0.68, respectively. The results of our experiments on different kernel sizes show that the F-1 score is the highest for a 3x3 kernel size with a score of 0.77 compared to a 6x6 kernel size with a score of 0.69. Additionally, the results of our experiments on different activation functions show that the F-1 score is the highest for the tanh activation function with a score of 0.77 compared to the ReLU and sigmoid activation functions with scores of 0.72 and 0.68, respectively. These results suggest that for our CNNLSTM2D model, a filter size of 3, a kernel size of 3x3, and a tanh activation function in the first layer provide the best performance.

Second Layer

In the MaxPooling3D layer, our experiments are summarized in Table 4. By analyzing the results of these experiments, we found that the best performing MaxPooling3D layer had a width dimension range of 5 to 10 and a height dimension range of 5 to 10, with a fixed depth dimension of 10. Table 12 shows the F-1 score obtained for different values of the width and height dimensions.

Based on the results, it is evident that the best F-1 score was obtained with a MaxPooling3D layer that had a width dimension of 10, a height dimension of 10,

Width	Height	F-1 Score
5	5	0.72
6	6	0.74
7	7	0.76
8	8	0.78
9	9	0.79
10	10	**0.81**

Table 12: Results of experiments on different dimensions of MaxPooling3D layer.

and a depth dimension of 10. This layer yielded an F-1 score of 0.81, which is the best score obtained for the MaxPooling3D layer in our experiments.

Third Layer

In this section, we present the results of our experiments on the impact of different dropout regularization rates on the performance of the TimeDistributed layer in our model.

Table 13 summarizes the results of our experiments on different dropout regularization rates. As shown in the table, the F-1 score is the highest for a dropout regularization rate of 0.3.

Dropout Regularization Rate	F-1 Score
0.2	0.81
0.3	**0.82**
0.4	0.80
0.5	0.78

Table 13: Results of experiments on different dropout regularization rates.

Fourth Layer

For the fourth layer, table 14 shows the results of our experiments. We observed that using 6 filters instead of the previously optimal 3 filters resulted in an increase in the F-1 score to 0.83, which is the highest score obtained for this layer.

Filter Size	F-1 Score
1	0.68
3	0.71
5	0.78
6	**0.83**

Table 14: Results of experiments on different filter sizes for the fourth layer.

Note that the 5th and 6th layers yielded identical results of the 2nd and 3rd layer.

Dense Layer

Based on our experiments, we found that the ReLU activation function yielded the best results for the dense layer in our CNNLSTM model. The number of neurons that produced the best performance was 512. Additionally, we found that using two dense layers performed better than using one or three dense layers. Table 15 shows the F1-scores obtained from each combination of hyperparameters.

Activation	**No. of Neurons**	**No. of Dense layers**	**F1-score**
ReLU	512	1	0.78
Sigmoid	512	1	0.73
Tanh	512	1	0.75
ReLU	1024	1	0.76
ReLU	512	2	**0.83**
ReLU	512	3	0.80

Table 15: Dense layer experiment results

Based on the above results, we chose to use a dense layer with ReLU activation function, 512 neurons and two dense layers for the best performance. The F1-score obtained in this configuration was 0.82.

In summary, the extensive experimentation carried out has led to the development of our most efficient Convolutional Neural Network - Long Short-Term Memory 2D (CNNLSTM2D) model. The model achieved a remarkable F-1 score of 0.83, demonstrating its exceptional performance in classification tasks.

Our experiments were designed to explore various model architectures, hyperparameters, and training techniques to optimize the CNNLSTM2D model's performance. We started with a basic CNNLSTM2D model and iteratively modified its architecture to improve its classification accuracy.

The result of our experimentation was a CNNLSTM2D model that outperformed all other models developed during the project. Its F-1 score of 0.83 shows that the model has an excellent balance of precision and recall, making it suitable for a wide range of classification tasks.

In conclusion, our experimentation has demonstrated the effectiveness of the CNNLSTM2D model in classification tasks, and we believe that this model can be further optimized for more complex tasks. These findings are not only relevant to the field of machine learning but also to other domains that require the classification of complex data. We hope that our research can inspire future studies that explore the potential of deep learning in solving real-world problems.

Transformer Model

We conducted experiments on our transformer model by tuning its hyperparameters to obtain the optimal performance. Table 6 shows the variations of the hyperparameters that we tested, which resulted in a total of 1024 variations.

After testing all the variations, we found that the best-performing hyperparameters were 8 transformer layers, 8 number of heads, 128 embed dim, 1e-4 layer norm eps, and 0.3 dropout rate, which yielded an F-1 score of 0.89.

This result shows that the transformer model is highly sensitive to its hyperparameters, and careful tuning can significantly improve its performance. The table 16 below shows the F-1 scores for the top 5 tested hyperparameter variations.

variation	F1-score
8,32,128,1e-6,0.3	0.88
8,32,128,1e-4,0.3	0.83
8,8,128,1e-3,0.2	0.85
8,16,128,1e-4,0.3	0.85
8,8,128,1e-4,0.3	**0.89**
8,8,128,1e-4,0.5	0.80

Table 16: Transformer Model F-1 Scores

As we can see from the table, the F-1 score steadily increases with the number of transformer layers and heads. The best performing hyperparameters were 8 transformer layers, 8 number of heads, 128 embed dim, 1e-4 layer norm eps, and 0.3 dropout rate, which yielded an F-1 score of 0.89. These results show that the transformer model is highly effective in capturing the semantic meaning of text and can achieve state-of-the-art performance with carefully tuned hyperparameters.

Results summary

Based on the results of the experiments conducted on both the transformer and the CNNLSTM models, it is evident that the transformer model outperformed the CNNLSTM model in terms of classification accuracy. The transformer model was able to achieve an F-1 score of 0.89, which is significantly higher than the F-1 score of 0.83 achieved by the CNNLSTM model.

There are several possible reasons why the transformer model outperformed the CNNLSTM model. One possible reason is that the transformer model is better suited for processing sequential data, which was the type of data used in this study. The transformer model utilizes the self-attention mechanism, which allows it to capture long-range dependencies between frames in a video. In contrast, the CNNLSTM model uses convolutional layers to extract local features and LSTM layers to capture sequential dependencies. While the LSTM layers are capable of capturing long-range dependencies, the convolutional layers are limited to extracting local features.

Another possible reason why the transformer model outperformed the CNNLSTM model is that the transformer model is more robust to hyperparameter tuning. The transformer model has several hyperparameters that can be tuned, such as the number of layers, the number of heads, and the dropout rate. In contrast, the CNNLSTM model has fewer hyperparameters that can be tuned, such as the number of convolutional layers, the number of LSTM layers, and the dropout rate. The results of our experiments showed that the transformer model is highly sensitive to its hyperparameters, and careful tuning can significantly improve its performance.

In conclusion, the results of our experiments suggest that the transformer model is better suited for processing sequential data than the CNNLSTM model. The transformer model was able to achieve a higher classification accuracy than the CNNLSTM model, and its performance was more robust to hyperparameter tuning. As such, the transformer model may be a better choice for our ASL recognition tasks; However, it is important to note that the performance of the transformer model may vary depending on the type of data and the specific task being performed.

5 Conclusion

The conclusion of any research work is critical in providing a summary of the findings, limitations, future work, and ethical concerns of the study. This section serves as a final analysis of the results obtained from the research, highlighting the implications of the research question and its contributions to the field. In this thesis, we explored the use of deep learning techniques to recognize American Sign Language (ASL) gestures. We have discussed our key findings, including the superiority of the transformer model over the CNNLSTM model and the importance of hyperparameter tuning. Additionally, we have identified the limitations of our research and the need for future work in improving the accuracy of the model. Furthermore, ethical concerns surrounding the use of deep learning models in ASL recognition were also discussed. In this section, we provide a comprehensive discussion of the key findings, limitations, future work, and ethical considerations of our research on ASL recognition using deep learning techniques.

5.1 Key Findings

This thesis aimed to investigate the use of deep learning techniques for American Sign Language (ASL) recognition. The research question was whether a deep learning model can achieve high accuracy and efficiency in ASL recognition. The two deep learning models investigated were Convolutional Neural Network-Long Short-Term Memory (CNNLSTM) and Transformer models. The results showed that the Transformer model outperformed the CNNLSTM model in terms of accuracy and efficiency. The Transformer model achieved an F1-score of 0.89, while the CNNLSTM model achieved an F1-score of 0.83. Additionally, the Transformer model achieved higher accuracy and efficiency with fewer parameters compared to the CNNLSTM model.

Notably, the findings show that deep learning techniques can effectively be used to improve the accuracy and efficiency of ASL recognition systems. It is worth noting that the results of this study have surpassed the outcomes of previous research papers in the same field, demonstrating the success of the Transformer model in achieving high accuracy and efficiency in ASL recognition, while dealing with a bigger ASL dataset compared to previous work. These results are of great importance to the deaf and hard-of-hearing community, as they can greatly enhance communication and accessibility in various settings. The success of the Transformer model serves as a promising avenue for future research, providing a benchmark for further advancement in ASL recognition technology.

The findings of this research hold great significance for the development of effective ASL recognition systems using deep learning techniques. The use of such systems can have a profound impact on the lives of individuals in the deaf and hard-of-hearing community, providing them with improved accessibility and communication. With the growing importance of technology in our daily lives, the need for efficient and accurate ASL recognition systems has become increasingly critical. By using deep learning techniques, we can address this need and develop systems that can facilitate communication and accessibility in a range of settings, including education, employment, and social interactions. The results of this research are an

important step towards creating such systems and providing support for individuals in the deaf and hard-of-hearing community.

5.1.1 Limitations

This research has brought to light a number of significant findings in the field of American Sign Language recognition, providing insights into the performance of deep learning techniques in this domain. However, like any research, there are some limitations that should be taken into account when interpreting the findings.

One of the most significant limitations of this research is the size of the dataset used. While the dataset used in this study is relatively larger than many other datasets used in ASL recognition research, it is still not sufficient to solve the bigger problem of recognizing all signs in the ASL vocabulary. This limitation could affect the generalizability of the findings and may limit the applicability of the models developed in this study to real-world ASL recognition scenarios. This research has only proved that deep learning algorithms could be furtherely explored and improved in order for us to find a better solution.

Another limitation of this research is that the models developed have only been tested on a single dataset, which was due to time constraints, limited technology available at our hands, and lack of diverse available ASL dataset. Thus, the performance of the models could be significantly different when applied to different datasets or real-world scenarios. Therefore, it is important to continue evaluating the models in a variety of contexts to determine their real-world utility and to build more diverse and representative datasets to enhance the generalizability of the findings.

In conclusion, this research has made significant contributions to the field of American Sign Language recognition, demonstrating the potential of deep learning techniques for improving the accuracy and efficiency of ASL recognition systems. Despite some limitations, the findings of this research suggest that continued research in this area has the potential to lead to significant advancements in ASL recognition technology, with significant implications for the deaf and hard-of-hearing communities.

5.1.2 Future work

Based on the conclusions drawn from the analysis of the CNN-LSTM and Transformer models for American Sign Language recognition, there are several avenues for future work that can build upon the current research and overcome its limitations.

Firstly, it is worth noting that the generalizability of our findings is limited by the size of the dataset used for training and evaluation of the models. To address this limitation, future work can focus on increasing the size of the dataset. This is a challenging task that requires a significant amount of time and resources. Collecting more diverse and extensive data that covers a broad range of signing styles, lighting conditions, and camera angles is necessary to ensure that the models can perform well in a variety of real-world scenarios. Such an effort will require collaboration with the deaf and hard-of-hearing community to collect data that reflects the diversity of their signing styles and experiences.

Another possible direction for future work is to investigate the potential of combining the CNN-LSTM and Transformer models for ASL recognition. Ensemble models that combine the strengths of both models can potentially improve the accuracy and robustness of the recognition system which has been proven through our work with CNNLSTM models. The CNN-LSTM can be used to extract spatial features from the input video frames, while the Transformer model can be employed to capture the temporal dependencies and long-range context information.

Furthermore, future work requires an exploration of the potential of transfer learning and domain adaptation techniques to enhance the performance of the ASL recognition models. However, this approach may require more advanced equipment and longer time frames for implementation. Transfer learning entails utilizing pre-trained models on similar tasks and fine-tuning them on the target dataset, whereas domain adaptation aims to adapt the model to new and unseen domains. By leveraging the knowledge learned from pre-trained models and adapting it to the ASL recognition task, transfer learning and domain adaptation can potentially improve the model's accuracy and generalization capabilities. The implementation of these techniques can aid in improving the generalizability and robustness of the model.

In addition, further research can focus on developing multimodal ASL recognition systems that integrate both video and accelerometer data. The accelerometer data captures the motion dynamics of hand movements, which can provide additional information to enhance the accuracy and robustness of the recognition system. Integrating multiple modalities can also potentially improve the user experience by making the recognition system more reliable and user-friendly. This approach has been shown to be effective in other areas of human activity recognition and has the potential to improve the ASL recognition system's overall performance. Additionally, this approach can be further enhanced by exploring new sensor technologies or combining multiple sensors to capture more comprehensive information about the signing movements. Such improvements will require advanced equipment and more extensive data collection and processing, highlighting the need for continued research in this area.

In summary, this research has shed light on the potential of deep learning techniques, specifically the CNN-LSTM and Transformer models, for ASL recognition. However, there are still several avenues for future work that can build upon the current research and overcome its limitations. By exploring these directions, researchers can potentially develop more accurate, robust, and generalizable ASL recognition systems that can benefit the deaf and hard-of-hearing community and enhance their communication capabilities.

5.1.3 Ethical concerns

As with any technological advancement, deep learning techniques for ASL recognition have raised ethical concerns that need to be addressed. This section will discuss some of the potential ethical concerns that arise from the use of ASL recognition technology and what can be done to mitigate them.

The first ethical concern of this technology is privacy. It is important to ensure that the users' data are kept private and secure. With the increasing use of deep learning algorithms in various fields, privacy concerns are becoming more apparent.

In the case of ASL recognition, the users' movements and signs are recorded, and this data can potentially be used for nefarious purposes. Therefore, it is important to ensure that strict data privacy policies are in place to prevent unauthorized access to the data.

Another ethical concern of ASL recognition is its potential impact on the deaf community. Some members of the deaf community view ASL as their primary language, and the use of this technology may be seen as a threat to their cultural identity. There is also the concern that the use of ASL recognition technology may lead to the erasure of ASL as a language, as people may rely more on technology to communicate than learning the language themselves. Therefore, it is essential to involve members of the deaf community in the development and implementation of ASL recognition technology to ensure that it is used in a way that is respectful of their language and culture.

Furthermore, the accuracy of ASL recognition technology may be affected by biases in the dataset used for training the model. For example, the dataset used to train the model may not have been diverse enough to capture the variability of ASL signs across different regions, leading to inaccuracies in recognition. Therefore, it is crucial to ensure that the dataset used for training is diverse and inclusive to minimize the potential for bias. We have worked our best to collect dataset that is inclusive of all genders; however, it was harder to find a dataset that is also inclusive of all races and ethnicities which is a problem that need to be further discussed.

Finally, there is the concern that the use of ASL recognition technology may lead to the replacement of human interpreters. While this technology may be a useful tool for communication, it should not be viewed as a replacement for human interpreters. Interpreters provide a valuable service to the deaf community, and their skills and expertise cannot be replicated by technology. Therefore, it is important to ensure that the use of ASL recognition technology does not lead to the displacement of human interpreters.

In conclusion, the development and use of ASL recognition technology have raised several ethical concerns that need to be addressed. It is essential to ensure that the technology is developed and implemented in a way that is respectful of the deaf community's language and culture, maintains user privacy, minimizes potential biases, prevents misuse, and does not displace human interpreters. By addressing these concerns, we can ensure that ASL recognition technology is used in a way that benefits the deaf community while upholding ethical standards.

References