Investigating the impact of fusion methods on emotion recognition performance

Bachelor Thesis

by

Kamila Datbayev

Reviewer: Prof. Dr. Dr.-Ing. Wolfgang Minker
Supervisor: M.Sc. Denis Dresvyanskiy

Institute of Communications Engineering
University of Ulm
02 May 2024

I certify that I have prepared this Bachelor Thesis by my own without any inadmissible outside help.

Ulm, 02 May 2024

(Kamila Datbayev)

Abstract

Recently, there has been a growing interest in multimodal emotion recognition. Using only one modality, such as speech, gestures, or facial features, often does not portray the complete emotional state because an emotion combines several modalities simultaneously. This work investigates the impact of various fusion methods on emotion recognition performance. The fusion methods incorporated two modalities: video with facial features and audio with audio features. This thesis provides insight into how features can be extracted from a dataset and further preprocessed for models based on individual modalities and an audio-visual fusion model, where data synchronization is crucial. The performance of video and audio models were evaluated and afterward, the fusion experiments were conducted. Each fusion method had a different architecture, but three of them incorporated a transformer. The experiments prove that even on challenging dataset like SEWA, the proposed fusion methods either outperform singular modality model or partly have the same results as them.
Overall, this work shows the importance of combining different modalities in emotion recognition task and presents promising results for future study in that area. Recent state-of-the-art studies show that emotion recognition can be done by utilizing multiple approaches leading to different results.

Contents

List of Figures vii

List of Tables ix

1. Introduction 1

2. Related Work and Background 3
2.1. State of the art of emotion recognition . . . 3
2.2. Machine Learning Algorithms . . . 5
2.2.1. Support Vector Machine and Support Vector Regression . . . 5
2.2.2. Decision Tree Classifier and Decision Tree Regressor . . . 9
2.2.3. Bayesian Ridge Regression . . . 12
2.3. Deep Learning . . . 14
2.3.1. Feed-Forward Neural Networks . . . 18
2.3.2. Convolutional Neural Networks . . . 22
2.3.3. Recurrent Neural Networks . . . 24
2.3.4. Transformer . . . 26

3. Data and Tools 31
3.1. AFEW-VA dataset . . . 31
3.2. SEWA dataset . . . 32
3.3. Feature Extraction . . . 33
3.4. Evaluation Metrics . . . 35

4. Experiments 37
4.1. Data Manipulation . . . 37
4.1.1. Data Loading . . . 37
4.1.2. Data Preprocessing . . . 37
4.2. Baseline . . . 38
4.2.1. Machine Learning Algorithms . . . 38
4.2.2. Neural Network . . . 40
4.3. Video Model . . . 42
4.4. Audio Model . . . 45
4.5. Audio-Visual Fusion . . . 45

5. Conclusion 51

A. Appendix 53

List of Figures

1.1. The valence-arousal space [100]. 1

2.1. 2D hyperplane visualized [78]. 6
2.2. 3D hyperplane visualized [78]. 6
2.3. Same hyperplane, but in 2D space [78]. 7
2.4. Linear SVR and Non-linear SVR after a transformation of the features into a higher-dimensional space via a kernel function [33]. 8
2.5. Basic decision tree structure [53]. 9
2.6. Example of a regression tree [52]. 10
2.7. Prediction space after dividing it into different regions [52]. 11
2.8. A 3D example plot of the prediction space [52]. 12
2.9. An example of standard linear regression compared to ridge regression. Blue represents training points and green represents test points [28]. . . . 14
2.10. An example of Bayesian Ridge Regression model compared to original data and OLS regression [10]. 15
2.11. Visualization of the regularization effect [89]. 16
2.12. The structure of an artificial neuron [69]. 16
2.13. Visualization of Tanh and ReLU activation functions [88]. 17
2.14. Visualization of a simple FFNN architecture [68]. 18
2.15. Visualization of a simple neural network model that goes through the back-propagation process [49]. 20
2.16. Example of 1D convolutional neural network architecture with two convolutional layers [80]. 22
2.17. Example of max pooling operation and average pooling with a 2x2 pixel filter size from 4x4 pixel input [97]. 23
2.18. Visualization of a simple RNN internal operation [82]. 24
2.19. Visualization of a GRU unit [82]. 25
2.20. The Transformer architecture [90]. 27
2.21. Visualization of the self-attention with key, query, and value transformations [40]. 28
2.22. Multi-Head Attention consists of several attention layers running in parallel [90]. 29

3.1. Distribution of arousal and valence values in AFEW-VA dataset [59]. . . . 31
3.2. Distribution of valence and arousal values in AFEW-VA, 2D plot [59]. . . 32
3.3. Distribution of arousal and valence values in SEWA dataset. 33
3.4. Distribution of valence and arousal values in SEWA, 2D plot. 34

3.5. The pipeline of the EfficientNet-B1 model [7]. 34
3.6. Visual representation of MAE using dummy data, where arousal MAE = 0.08 and valence MAE = 0.10. 35

4.1. Visual representation of the Feed Forward Neural Network used in the baseline experiments. The variables x in the input layer are batches that are processed sequentially one after the other. 41
4.2. Visual representation of the RNN-based video model used in the experiments. 43
4.3. Visual representation of fusion method 1. 46

A.1. Visual representation of fusion method 2. 53
A.2. Visual representation of fusion method 3. 58
A.3. Visual representation of fusion method 4, variation 1. 59
A.4. Visual representation of fusion method 4, variation 2. 60
A.5. Visual representation of fusion method 4, variation 3. 61
A.6. Data preprocessing in the baseline model. 62
A.7. Data preprocessing in the video model. 63
A.8. Data preprocessing in the audio-visual model. 64

List of Tables

4.1. Results of the used machine learning algorithms on AFEW-VA. In the following, in each table v = valence, a = arousal 39
4.2. Results of the used machine learning algorithms on SEWA. 40
4.3. Results for the baseline Feed Forward Neural Network on AFEW-VA . . . 41
4.4. Results for the baseline Feed Forward Neural Network on SEWA 42
4.5. Best results of the video-based model on the AFEW-VA dataset 44
4.6. Best results of the video-based model on the SEWA dataset 44
4.7. Performance of the audio-based model on the SEWA dataset 45
4.8. Results of the best fusion method on test data (SEWA dataset). 49
4.9. Results of the best fusion methods on development data. 49

A.1. Video-based model on AFEW-VA dataset 54
A.2. Video-based model on SEWA dataset 55
A.3. Fusion methods on SEWA dataset (test data) 56
A.4. Fusion methods on SEWA dataset (development data) 57

1. Introduction

Emotion plays a crucial part in everyday human experiences, impacting communication and interaction. The rapid growth of interest in human-computer interaction raises the need for systems that can successfully capture the human's emotional state [62][29].

Emotion states are usually represented using either a categorical or dimensional approach. In a categorical approach, Ekman [43] outlines six basic emotions (anger, happiness, fear, sadness, disgust, and surprise) based on a cross-cultural study. This study shows that humans, regardless of cultural background, perceive basic emotions in the same way. In the dimensional approach, emotions are represented through a continuous numerical value in multiple dimensions, like valence-arousal space. Valence represents the degree of positive and negative feelings, and arousal represents the degree of excitement and calm. Based on this valence-arousal space, every emotional state can be represented as a point in the coordinate plane, as shown in Figure 1.1 [100][57]. Praveen et al. [73] note that dimen-

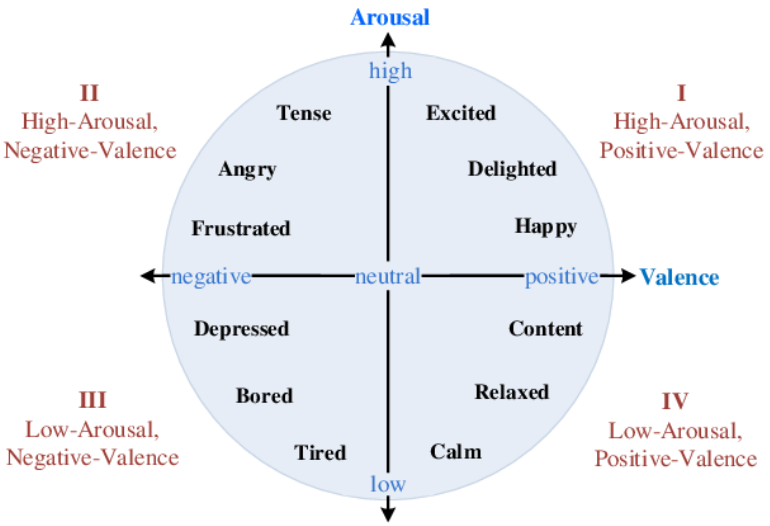

Figure 1.1.: The valence-arousal space [100].

sional modeling of emotions is generally more challenging than categorical one because it is more difficult to get a continuous scale of annotations than an emotion category. The annotations often seem noisy and ambiguous due to the continuous range of emotions. Several datasets were explicitly made for dimensional emotion recognition. Among them is SEWA [60], which is utilized in this work.

Emotions are usually portrayed nonverbally with facial expressions, gestures, body language, and verbally with speech [64]. In real life, emotions are expressed simultaneously with several nonverbal and verbal modalities. So, to capture more emotional context, a multimodal fusion of modalities can be used. In this work, we investigate the impact of audio-visual fusion methods on emotion recognition performance. First, each individual modality and its performance are examined. Afterward, audio-visual fusion experiments are conducted. The preprocessed audio and video data, which served as input for all models, were also discussed.

Moving forward, this paper covers everything described in the following. Chapter 2 summarizes related work and the background knowledge of machine learning algorithms and deep learning. Chapter 3 covers the used datasets in this work and how audio and video embeddings were extracted from the original data. Additionally, the evaluation metrics used in the experiments are described. Chapter 4 follows the explanation of how experiments were performed. First, the data manipulation process, including data loading and preprocessing for models based on one modality and fusion models, is demonstrated. After setting up the baseline with algorithms and a simple feed-forward neural network, models using only one modality were built, and their results were evaluated in sections 4.3 and 4.4. Lastly, four fusion methods were performed, experimenting with various window lengths and architectures. In the end, Chapter 5 follows a summary and reflection of this work. Also, possible future work was discussed. Appendix A contains tables with the results of experiments on video, audio, and fusion models. Additionally, the visualization of data preprocessing and fusion methods' architectures can be found there.

2. Related Work and Background

2.1. State of the art of emotion recognition

Emotions are a crucial aspect of human communication, and a significant portion of emotional expression is conveyed through non-verbal cues such as facial expressions, body language, and tone of voice. Video-based emotion recognition allows for analyzing these non-verbal cues, providing a more comprehensive understanding of an individual's emotional state [47]. Multiple studies were conducted using video information for emotion recognition.
In Gunavan et al.'s paper [47], emotion recognition using video was implemented by selecting the Indian Spontaneous Expression Database (ISED) for training. The process involved preprocessing and feature extraction using LeNet and AlexNet neural networks. Parameter tuning was conducted to enhance performance, resulting in AlexNet achieving 93.00% recognition accuracy with the Stochastic gradient descent (SGD) optimizer [47].

Xu et al. [96] implemented a video-based Emotion Recognition system using a pipeline consisting of four main modules: image processing, deep feature extraction, feature aggregation, and emotion classification. By using deep learning techniques such as CNNs, LSTM, and 3D Convolutional Networks (3D ConvNets), the system was able to effectively recognize emotions from videos by capturing both spatial and temporal information [96]. This approach proved highly effective, achieving an accuracy of 48.01% by weighting the scores from different models, surpassing the baseline method by 9.2% [96].

In the study of Qui et al. [74], Video Emotion Recognition was implemented using a Dual Focus Attention Network (DFAN), which consisted of two attention modules, Time Series Focus and Frame Objects Focus, to be able to effectively identify the most informative visual cues in the video on both temporal and spatial dimensions. The results of the implementation of the DFAN showed state-of-the-art performance on two benchmark video emotion datasets: VideoEmotion and Ekman [74]. The recognition accuracy was significantly improved compared to other approaches, achieving recognition accuracies of 53.34% on the VideoEmotion dataset and 57.37% on the Ekman dataset [74].

Unlike visual emotion recognition, audio emotion recognition is more challenging "due to the complexity of emotional expressions"[84]. However, speech emotion recognition using audio data alone can be effective and has shown promising results in various studies [84]. In Suganya et al.'s work [84], speech emotion recognition was implemented using an end-to-end deep learning approach that applied a deep neural network directly to raw audio

recordings of speech to learn high-level representations from the audio waveform. The CNN model with nine weight layers achieved an overall accuracy of 68.6% for IEMOCAP over four emotions and 85.62% for EmoDB over seven emotions. The results showed that the performance of the proposed model was better compared to traditional machine learning approaches and deep learning on the spectrogram of audio recordings in the same datasets [84].
Zheng et al. [101] implemented a speech emotion recognition system using an acoustic segment model (ASM) to segment speech data and a deep neural network (DNN) classifier for emotion classification. The results showed a weighted accuracy of 73.9% and an unweighted accuracy of 70.8% on the IEMOCAP dataset, outperforming state-of-the-art methods. Additionally, achieving accuracies of 58.5% for anger, 84.6% for neutral, 18.3% for happy, and 76.8% for sad emotions demonstrate a reasonable performance in distinguishing between different emotional states [101].

Koolagudi et al. [58] note that emotions are often expressed through multiple modalities, such as facial expressions, gestures, and tone of voice. Combining information from multiple modalities (e.g., audio and visual cues) can improve the accuracy of emotion recognition systems.
In addition, there is an observation in several research works of the past few years that the fusion of modalities for emotion recognition tasks gets better results than using only one modality [85, 48, 86, 51, 73]. In found studies, popular techniques are Convolutional Neural Networks (CNNs) or Transformers. Let us address CNNs first.

In the study of Sultana et al. [85], CNNs were utilized for both audio and visual modalities in the emotion recognition process. A one-dimensional CNN architecture was employed for audio data to learn complex patterns and representations from the audio domain. On the other hand, a three-dimensional CNN framework was used to analyze video frames and extract essential facial characteristics for video data. "The fusion technique involve[d] concatenating and extending the outputs of the audio and visual models." [85] The study's results indicate that the multimodal fusion model, combining audio and visual data for emotion recognition, achieved a respectable accuracy level of 66.90% [85].
Another study that used CNNs for a combination of audio and visual information was conducted by Haddad et al.[48]. CNNs were used for audio data with an AlexNet architecture and for visual data with a VGG-Face network to extract features for emotion recognition. Results showed that the fusion of all input types had an accuracy of 86.36%, demonstrating the effectiveness of the multimodal approach for emotion recognition [48]. The main difference to these works is the use of Transformer architecture instead of CNNs. Like Haddad et al. [48], we used the pooling methods within our fusion architectures.

However, researchers are increasingly using Transformer over CNNs, as it represents a newer approach that often yields better results. Let us now address Transformer.

Sun et al. [86] introduce a novel approach, Multimodal Cross- and Self- Attention Network

(MCSAN), to effectively fuse textual and acoustic information for accurate emotion recognition in speech. MCSAN addresses the challenge of integrating linguistic and acoustic information in Speech Emotion Recognition by employing parallel cross- and self-attention modules. Cross and self-attention are attention mechanisms in Transformer architecture that mix two different embedding sequences; they will be explained more in-depth later in the Background chapter. Licai Sun et al. [86] achieved significantly better performance when fusing linguistic and acoustic information than using either modality alone for Speech Emotion Recognition. When comparing the performance of MCSAN with and without audio or text inputs, the model achieved higher accuracy and effectiveness when both modalities were combined, showcasing the value of multimodal fusion in improving emotion recognition. The difference to the work of Sun et al. is the use of video instead of textual information.

Huang et al. [51] focused on model-level fusion, which utilizes the Transformer model to attend to interactions between audio and visual modalities and combine it with Long short-term memory (LSTM), which focuses on capturing temporal context information effectively. This combination allows for more effective integration of audio-visual modalities information on the model level fusion, leading to improved results in continuous emotion recognition tasks. The difference to this work is using only Transformer's attention mechanisms, rather than combining those with LSTM. Besides Conv1D, other pooling methods like AVG Pool, MAX Pool were used as well in this thesis, which will be presented later in Chapter 4.

The research by Praveen et al. [73] focused on enhancing facial and vocal modalities extracted from videos to predict valence and arousal in emotion recognition. The proposed model combined audio (A) and visual (V) modalities using a cross-attention method to understand how they relate to each other. This way, the fusion model achieved higher accuracy and robustness in capturing emotional cues than individual modalities alone [73]. The study also explored self-attention and cross-attention fusion based on transformers.

Overall, this work uses the known attention techniques in Transformer, self-, and cross-attention in audio-visual fusion. However, it is conducted on a different dataset and uses other fusion approaches than those in the above studies.

2.2. Machine Learning Algorithms

2.2.1. Support Vector Machine and Support Vector Regression

Support Vector Regression (SVR) is a machine learning technique for regression problems, aiming to minimize the generalization error bound [38]. SVR is a variant of Support Vector Machine (SVM) algorithm.

Typically used in classification, SVM seeks to identify a hyperplane that effectively sep-

arates data points into distinct classes, maximizing the margin between the hyperplane and the nearest data points of each class while minimizing classification errors. When new data points need to be classified, the hyperplane acts as the decision boundary [78].

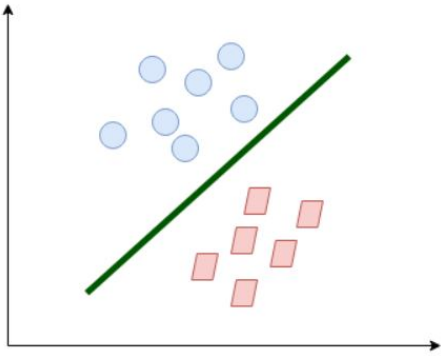

Figure 2.1.: 2D hyperplane visualized [78].

Figure 2.1 shows the assignment to a category divided by the hyperplane. This data is called linearly separable.

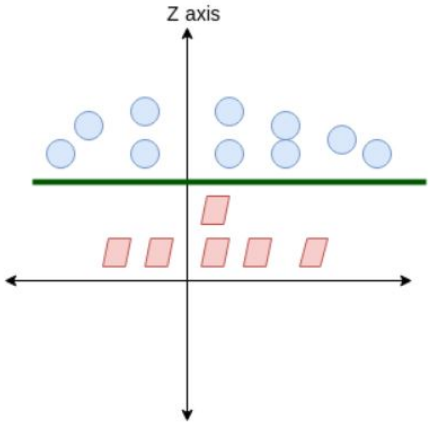

Figure 2.2.: 3D hyperplane visualized [78].

Sometimes, the data is more complex, and a simple line for class separation is not possible. Like in Figure 2.2, the classes can be separated using a new dimension: the z-axis. After this transformation, the data is linearly separable in three dimensions.

After transforming the graph back into 2D space, the hyperplane looks like a circle in Figure 2.3

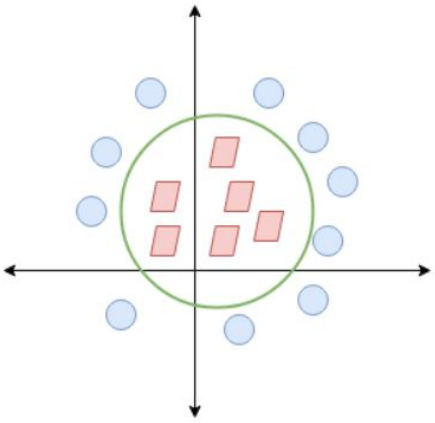

Figure 2.3.: Same hyperplane, but in 2D space [78].

In practical scenarios, when dealing with datasets containing numerous features, applying transformations involving multiple polynomial combinations of these features can result in unnecessary computational expenses. This is because operations with the higher-dimensional vectors in the transformed feature space are required to train a support vector classifier and optimize the objective function.
The solution to this problem is the so-called "kernel trick". It is a technique that allows to avoid the need to explicitly convert the data into a higher-dimensional space. Instead, kernel functions focus on comparing the original data observations based on their similarities.

Kernel function takes $\mathbf{x}$, $\mathbf{z}$ as input vectors in the original space and returns a scalar that represents the inner product of vectors in a potentially much higher-dimensional space.
If we have data $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ and a map $\phi : \mathcal{X} \to \mathbb{R}^N$ then $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ is a kernel function [67][95].

So this way, it computes these inner products without ever having to map the data points into the higher dimensions, which does not require a high computational effort. Naturally, all these calculations happen inside the kernel functions when implementing it [30].
Linear kernel is the simplest kernel form, used when the data is linearly separable. In SVR, it effectively means no transformation is needed, and the original feature space is used. The equation looks like $k(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z}$.
Other popular kernels are polynomial, radial basis function (RBF), and sigmoid [70].

In SVR, on the other hand, the hyperplane represents the regression line that minimizes the error between the predicted and actual output values while maximizing the margin. The hyperplane is determined by support vectors, which are the data points closest to the hyperplane and significantly influence the model, while other data points have little or no impact [91]. So, in this case, the output is not a categorical label; it is a continuous

value representing the predicted target variable for each data point [92][78].
In emotion recognition tasks, arousal and valence are essential components typically evaluated on continuous scales [41]. Therefore, SVR is well-suited for such scenarios as it involves estimating a position along a continuous range, making it a more appropriate option.

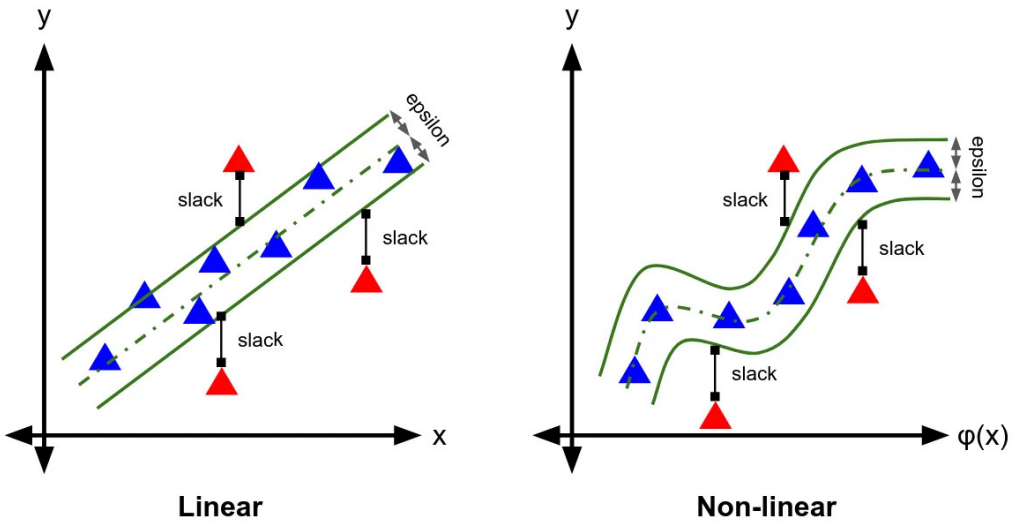

Figure 2.4.: Linear SVR and Non-linear SVR after a transformation of the features into a higher-dimensional space via a kernel function [33].

Figure 2.4 shows Linear SVR and Non-linear SVR. Each graph represents the concept of fitting an SVR model to data points marked with triangles and squares. The goal is to find the function that best fits the data while allowing for some errors within a certain margin. The following parameters are displayed:

- Epsilon (ϵ): the width of the margin around the predicted function. Data points within this margin are considered acceptable.
- Slack Variables (ξ): the distances by which the data points exceed the epsilon margin. The goal is to minimize the slack variables, which means minimizing the errors outside the margin [92].

In practice, Grid Search Cross-Validation (GridSearchCV) inside SVR can be used in emotion recognition. This method works through multiple combinations of parameter tunes to determine which tune gives the best performance. Each combination of C and kernel will be tried in a defined parameter grid. The regularization parameter C determines the trade-off between achieving a low training error and a low testing error (overfitting). The kernel parameter selects the type of kernel to be used in the algorithm. Once GridSearchCV has tested all parameter combinations, it will identify and return the model with the best performance metrics. This model is then used to predict arousal or valence on the development and test datasets [31].

2.2.2. Decision Tree Classifier and Decision Tree Regressor

Decision tree is a machine learning algorithm used for classification and regression tasks. It uses the features within a dataset to progressively split it into smaller, more manageable subsets designed to minimize impurity in each resulting subset [39][71].

Decision Tree Classifier deals with categorical target variables. The classifier traverses from the tree's root to a leaf node that matches the features of the new input data. The predicted category is typically the one that represents the majority of the data points in that leaf node.

Decision Tree Regressor (DTR), on the other hand, is used to predict continuous target variables [46][71]. Since arousal and valence are typically measured on a continuous scale, indicating degrees of emotional intensity and positivity/negativity [41], and a regressor is designed to predict such continuous outcomes, DTR seems more suitable for the emotion recognition task. Therefore, this subchapter will focus on the decision tree regressor.

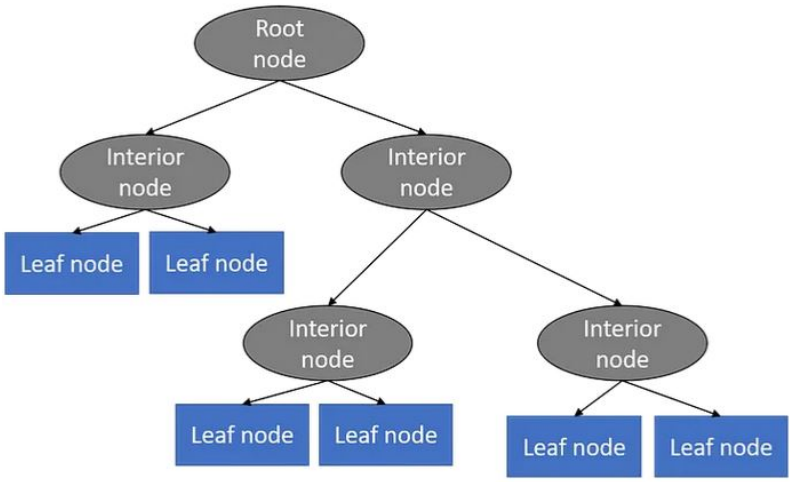

Figure 2.5.: Basic decision tree structure [53].

As seen in Figure 2.5, the decision tree's structure consists of nodes, branches, and leaves. The main node in a tree is known as the root node, from which the tree branches out into interior nodes and leaf nodes. Each interior node represents a decision point and splits into two child nodes, making the tree a binary tree [79][53].

The growth of a regression tree involves dividing the predictor space by taking the entire set of feature variables $(X_1, X_2, \ldots, X_p)$ and splitting it into several distinct and non-overlapping regions $(R_1, R_2, \ldots, R_j)$. First, recursive binary splitting is used to construct the regions $R_1, \ldots, R_j$. For each potential split, the Residual Sum of Squares (RSS) is

calculated to measure the effectiveness of the split as follows:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \tag{2.1}$$

where R_j represents one of the regions created by the split, y_i are the actual target values in that region, and $\hat{y}_{R_j}$ is the average of target values within region R_j.

The goal is to find the split with the lowest total RSS to reduce errors in the model. This means taking a feature and a specific split point where the differences between the actual and average values in each group are smallest. For any feature j and a split point s, the data is divided into two groups:

$$R_1(j, s) = \{X \mid X_j < s\} \quad \text{and} \quad R_2(j, s) = \{X \mid X_j \geq s\} \tag{2.2}$$

The algorithm selects the split that minimizes the errors in predictions for the two resulting regions:

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \tag{2.3}$$

After the best split is chosen, the process is repeated for each new region created by the previous split. This continues until a stopping criterion is reached, like a minimum number of data points per region [52].

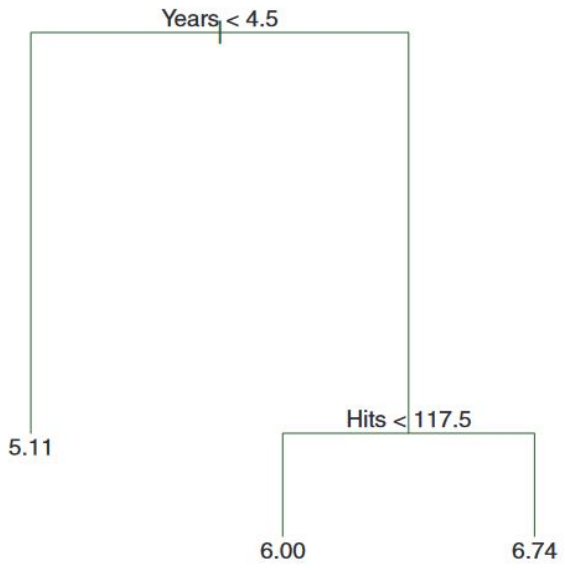

Figure 2.6.: Example of a regression tree [52].

Figure 2.6 shows an example regression tree for predicting the log salary of baseball players based on the number of years they have played in the major leagues and the number of hits they made in the previous year. The tree structure is displayed with nodes indicating the decision rules and leaves showing the average log salary for players falling into each category.

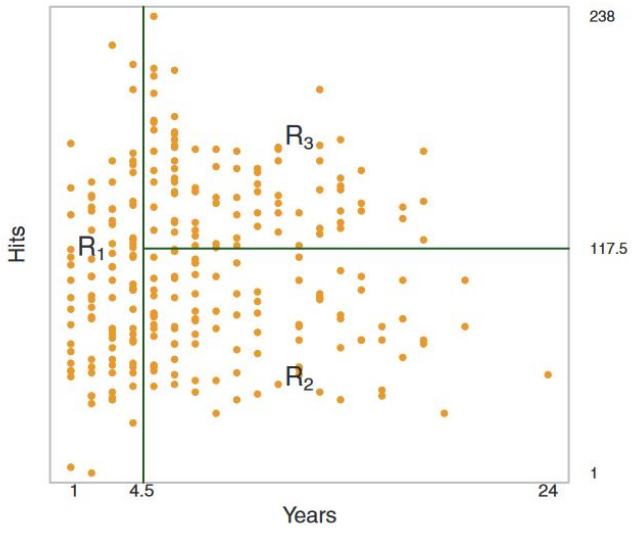

Figure 2.7.: Prediction space after dividing it into different regions [52].

Figure 2.7 visually represents how the prediction space is divided into different regions based on the splits defined in the regression tree. Each region (R_1, R_2, R_3) corresponds to a combination of conditions from the tree's nodes, and the average salaries for these regions are graphically shown in a two-dimensional space of 'Years' and 'Hits.'

In typical representations of the prediction space, especially in 2D plots, the target variable that the model needs to predict is not visible. Such plots display only the input features, and showing the target variable would require a 3D representation to capture the relationship between the features and the target. Figure 2.8 provides an example of how a prediction space can be visualized along with the target variable Y [52].

After a tree has been grown, it may be pruned back to prevent overfitting. Pruning involves removing subtrees from a fully grown tree to improve the tree's predictive ability on unseen data [52]. After the decision tree is built, predictions can be made by guiding a new data point through the tree down to the correct leaf. Once the data point reaches a leaf, the predicted value is just the average of all the training data values that also end up in that leaf [83].

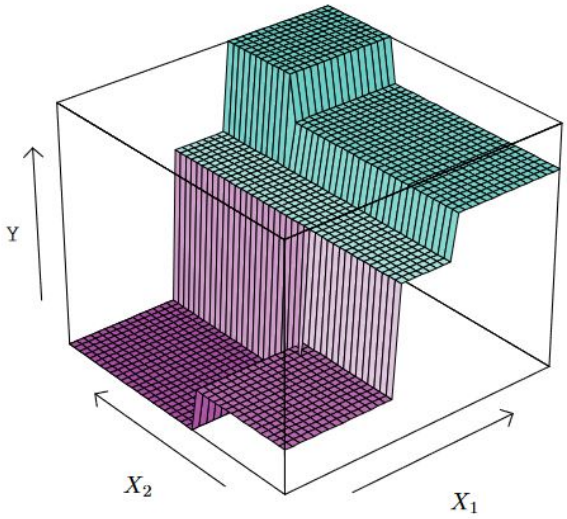

Figure 2.8.: A 3D example plot of the prediction space [52].

2.2.3. Bayesian Ridge Regression

Bayesian Ridge Regression is a machine learning algorithm that predicts outcomes in regression tasks [10]. As the name suggests, it consists of Bayesian inference and Ridge Regression. Both terms will be explained first to understand the algorithm.

Bayesian inference (BI) is a statistical method that allows to make better predictions and decisions by using prior knowledge and new evidence. It starts with a prior distribution, the initial belief about the parameters before seeing the data. This could represent assumptions or existing knowledge about a situation [93].
When new data is observed, Bayesian inference doesn't just consider this new data alone. Instead, it combines the new data with the prior beliefs to form the posterior distribution. This posterior gives a new and improved understanding of the parameters after taking into account the new evidence [93]. The BI procedure can be broken down into three main steps:

- Prior Distribution: Defining the prior distribution that reflects the initial belief about the parameters. This can be based on previous studies, expert opinions or assumptions.

- Likelihood: Considering the likelihood, which is the probability of observing the data given particular values of the parameters. This part focuses only on the new data.

- Posterior Distribution: Combining the prior distribution with the likelihood using Bayes' Theorem to compute the posterior distribution. This step mathematically blends prior beliefs and the evidence from the data (likelihood) to update the understanding of the parameters [93].

Bayes' Theorem for probability distributions is often stated as:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}, \tag{2.4}$$

where the symbol "$\propto$" means "is proportional to". This formula implies that the posterior distribution is proportional to the likelihood of the observed data multiplied by the prior distribution [42].

Ridge Regression (RR), also known as Tikhonov or L2 regularization, is a technique that deals with multicollinearity[1] [36][72][28][89].
RR is an extension of ordinary least squares (OLS) regression, a technique for finding the best-fitting line through a set of data points by minimizing the differences between the observed values and the values predicted by the line [14]. Upon that, a penalty term is added to the OLS loss function to shrink the regression coefficients toward zero, which can lead to more stable and reliable estimates [94][89].

The objective function for ridge regression is:

$$\text{minimize} \quad \underbrace{\|Y - X\beta\|^2}_{\text{Loss}} + \underbrace{\lambda\|\beta\|^2}_{\text{Penalty}}, \tag{2.5}$$

where Y is the dependent variable, X is the matrix of independent variables, β is the vector of coefficients to be estimated and λ is the tuning parameter that controls the amount of shrinkage.

Figure 2.9 shows that by introducing the penalty term and changing the slope of the line, ridge regression does not fit the training data as closely as the standard regression line anymore, but it highly improves the prediction on test data, therefore the test error is lower.

Altogether, Bayesian Ridge Regression (BRR) is a variation of the Ridge Regression with a difference of applying probability to the regression problem. Instead of finding the single "best" value of the model parameters, BRR considers a range of possible models and weights them according to their probability given the existing data [76][55]. It combines prior beliefs about the values (the priors) with the evidence provided by the data (the likelihood) to form an updated belief (the posterior) [76]. It is obvious that it is the Bayes' Theorem, which was already explained above.
The key point is that in Bayesian Ridge Regression, the parameters are not fixed values but distributions. This allows to account for the uncertainty in the predictions: BRR

[1] It means that the predictors in a regression model are linearly dependent [13].

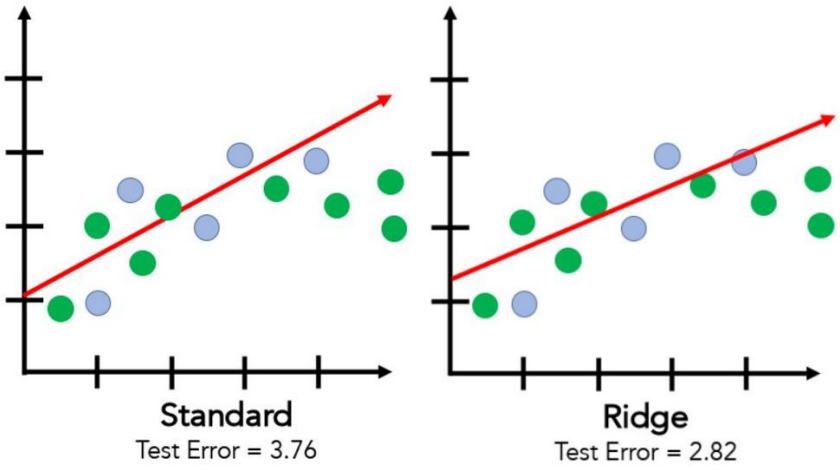

Figure 2.9.: An example of standard linear regression compared to ridge regression. Blue represents training points and green represents test points [28].

not only determines the placement of the line but also the level of confidence in different segments of the line. The main idea of BRR is that the model learns and adapts as it sees more data, becoming less wrong over time.

Figure 2.10 shows different estimates for the weights of the features in a Bayesian Ridge Regression model. The green line represents the estimates given by the BRR, which are generally more steady than the standard OLS regression, shown in blue. The yellow line represents the actual weights. Since Bayesian Ridge Regression avoids swinging to extremes, as seen in the OLS estimates, it helps make the model more reliable and a "good fit" [76], as shown in Figure 2.11 too.

Bayesian Ridge Regression also seems suitable for the emotion recognition task when predicting arousal and valence values since it naturally includes regularization, which helps to prevent overfitting, as stated earlier. Emotion recognition data is often considered uncertain [81] and Bayesian inference is good at handling such uncertainty by providing a distribution of possible outcomes, not just a single prediction, as also stated before. The data can also often contain extreme values [56] and BRR is, in this case, beneficial since it is more robust to outliers[2] as it was brought in an example in Figure 2.10.

2.3. Deep Learning

Deep learning (DL) is a subset of machine learning (ML) that has become a widely used computational approach in the field of ML due to its impressive results. One of the ben-

[2]Data points that differ significantly from other observations [15].

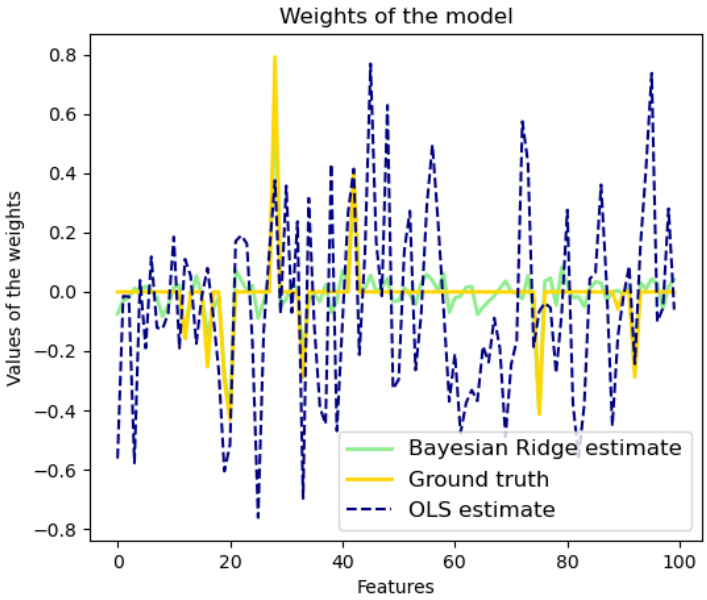

Figure 2.10.: An example of Bayesian Ridge Regression model compared to original data and OLS regression [10].

efits is the ability to learn massive amounts of data, which is why it is applied in various application areas, including emotion recognition [77][54][35].
DL is based on Artificial Neural Networks (ANN) [61]. Artificial neural networks are popular machine learning techniques that simulate the mechanism of learning in biological organisms. The neurons in the nervous system are connected to each other through synapses, using axons and dendrites. Living organisms learn by responding to external stimuli, which can result in changes to the strength of synaptic connections [61].

In ANNs, computational units which represent the neurons, are interconnected through weights, analogous to synaptic strengths in biological neurons. Each neuron receives inputs that are scaled by weights, influencing the function that the neuron computes. The network processes these inputs by transmitting the resulting values from the input neurons through to the output neurons, using the weights as modifying factors. Learning in these networks involves modifying these weights based on training data, which consists of input-output pairs. For example, in emotion recognition, features and their corresponding labels (arousal and valence) train the network. In this case, features are the input and the labels are the outputs. The network adjusts weights in response to prediction errors with the goal to modify the computed function to make the predictions more correct in future predictions. This process, called model generalization, allows the network to apply learned patterns to new, unseen data [34].

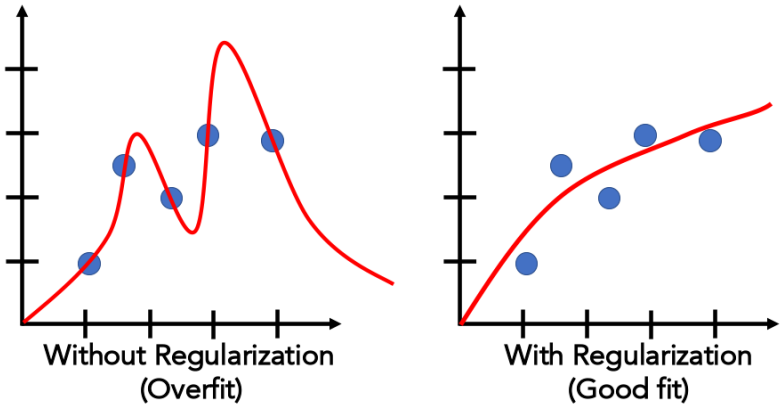

Figure 2.11.: Visualization of the regularization effect [89].

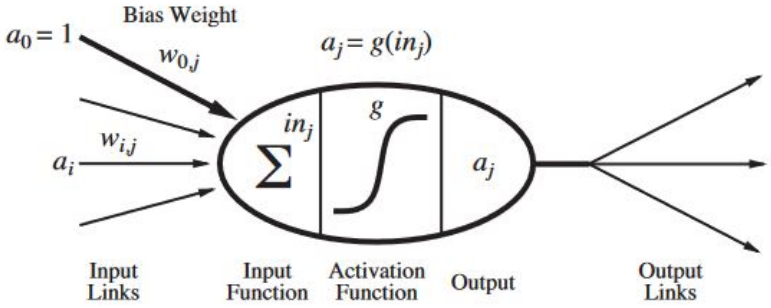

Figure 2.12.: The structure of an artificial neuron [69].

The Figure 2.12 demonstrates a simple mathematical model of the neuron. Let us break down what each part of the neuron model means [69]:

- Input Links: These are the connections from the previous layer's neurons to the current neuron. Each connection has an associated weight $w_{i,j}$, which determines the strength of the signal passed to the neuron. a_i is the activation from a neuron in the previous layer. It is the output of neuron i that is being fed into neuron j. Bias Weight ($a_0 = 1$ and $w_{0,j}$) effectively has the role of shifting the activation function to enable accurate responses, independent of other inputs.
- Input function: The function sums all weighted inputs, including the bias weight. The formula is expressed like this:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i, \tag{2.6}$$

  where in_j is the input to the neuron before the activation function is applied.

- Activation Function: It takes the input in_j and transforms it into the output activation a_j.

- Output: This is the result after the activation function g is applied to the weighted sum of inputs in_j. The output activation for this neuron is then used as input to neurons in the next layer or as part of the final output if this is the output layer. The output is calculated as:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right) \tag{2.7}$$

- Output Links: These are the connections that transmit the activation to the neurons in the next layer. Just like input links, each output link will have its own weight that will adjust the signal for the next layer's processing.

There are different activation functions that can be applied, like Rectified Linear Unit (ReLU), Tanh function, threshold function and logistic function [69]. Since only ReLU and Tanh functions are used in the experiments in this work, we will mainly focus on them. The ReLU function is a widely used activation function in deep learning, because

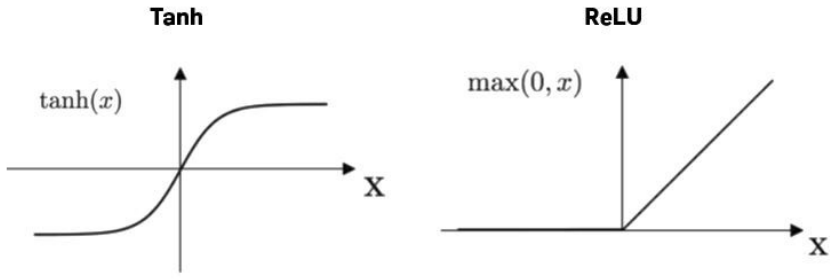

Figure 2.13.: Visualization of Tanh and ReLU activation functions [88].

it has the advantage of not activating all the neurons at the same time. The neurons will be deactivated only if the output of linear transformation is less than zero. In Figure 2.13 you can see that any negative input values result in a zero and any positive value x returns the value back. The function can be written as [8]:

$$f_{\mathrm{ReLU}}(x) = \max(0, x) \quad \text{or} \quad f_{\mathrm{ReLU}}(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.8}$$

Another commonly used activation function is tangent hyperbolic or simply *tanh* . It is a s-shaped curve that maps any input to a value between -1 and 1. In this work, the tanh function is used in the output layer to constrain the output to between -1 and 1, which

makes sense in emotion recognition tasks where arousal and valence are predicted, which typically have normalized ranges [44]. The function is defined as [88]:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.9}$$

After deciding on the mathematical model for individual neurons, the next step is connecting them together, which forms a network [69]. There are several types of networks in Deep Learning like Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Transformers that will be explained later on in this Chapter.

2.3.1. Feed-Forward Neural Networks

The most basic architecture is feedforward neural network (FFNN) or multilayer perceptron (MLP). The goal is to approximate the function $y = f^*(x)$, which maps input x to output y. For example, in emotion recognition function f^* maps input features x (such as facial expressions or physiological signals) to emotional states like arousal or valence (output y). FFNN defines a mapping $y = f(x; \theta)$ and learns the values of the parameters θ through training processes. The idea is to adjust θ so that the network's predicted mapping $f(x; \theta)$ comes as close as possible to the true function f^* [45].
The name 'feedforward'emerges from the computation flow in just one single direction - from the input x, through intermediate layers, to the output y. There are no feedback connections, which means that the output of the model is not fed back into itself at any point [45].

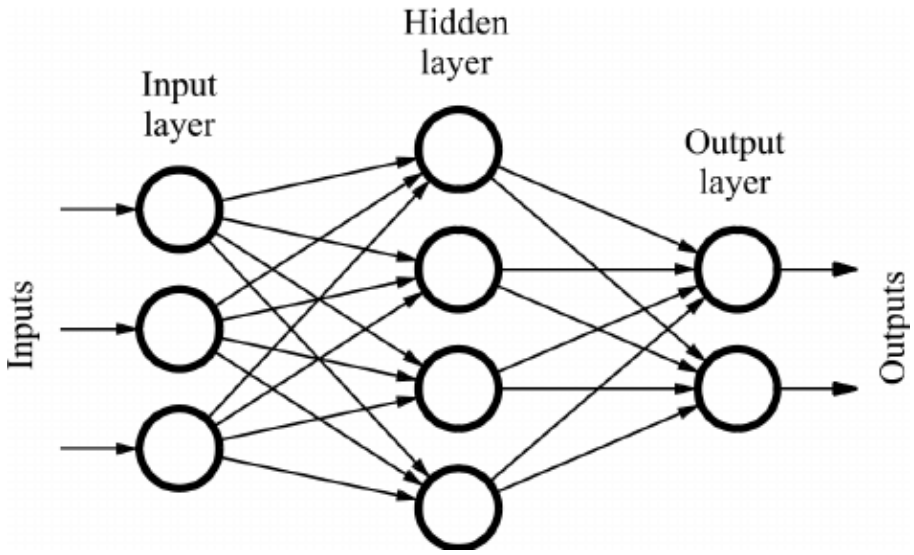

Figure 2.14.: Visualization of a simple FFNN architecture [68].

Figure 2.14 demonstrates a simple FFNN architecture. Each circle represents a neuron, and the lines between neurons represent the connections across which the signals travel. The Figure 2.12 from before that demonstrated the structure of an artificial neuron is a 'zoom in' on an individual neuron, which is one of the circles in Figure 2.14. FFNN consists of three layers:

- Input Layer: This layer accepts the data (typically in the form of a vector) and passes it to the next layer without performing any computation [3].
- Hidden Layer: Each neuron in a hidden layer receives inputs from all the neurons in the previous layer, multiplies these inputs by its weights, and then passes the result through an activation function. The output of each neuron is then used as input to the next layer [26].
- Output layer: This layer is the last layer of the network, and its neurons represents the predictions of the model [3].

To sum it up, when the feedforward network accepts an input x and produces an output y, information flows forward through the network. The process of inputs x providing the initial information that then propagates up to the hidden units at each layer to produce y, is called *forward propagation* [45].

The 'Multilayer' part in the MLP comes from the fact that this type of network includes one or more hidden layers. For comparison, a single-layer neural network has no hidden layers, only an input and an output, and is called a perceptron. So as soon as one hidden layer is added, it can be considered a multilayer network [69]. It is important to note that even though the above neural network has one hidden layer, it is not automatically considered a deep neural network. Networks that have multiple (at least two) hidden layers are classified as deep neural networks. These networks are part of deep learning because they have the 'depth' to represent more complex hierarchical features [77].
Nevertheless, the above FFNN incorporates the principles of artificial neural networks, which are the foundation of modern deep learning, which is why it is included in this Chapter.

Training a neural network is an important phase in developing a deep learning model. The training process involves iterative forward and backward passes through the network to adjust the weights and minimize prediction error. During forward propagation, an input x is passed through the network, producing an output [49].

Let us say, there is a model that takes an input x and maps it through two functions f_1 and f_2 associated with weights w_1 and w_2, respectively. The output of the first function, h_1, serves as the input to the second function, which produces the predicted output $\hat{y}$. It can be represented as $h_1 = f_1(w_1 \cdot x)$ and $\hat{y} = f_2(w_2 \cdot h_1)$. The loss is computed by the function loss $= L(f_2(w_2 \cdot f_1(w_1 \cdot x)), \hat{y}) = L(\hat{y}, y)$, which measures the difference between the predicted output $\hat{y}$ and the true output y. The model's goal is to adjust weights in

order to reduce this loss, which represents the error in prediction [49].

Now begins the process of *backpropagation*, where the model learns. To update the weights w_1 and w_2, we calculate the gradients of the loss function with respect to each weight. Using the chain rule, the gradient with respect to w_2 is represented as:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}, \tag{2.10}$$

where $\frac{\partial \hat{y}}{\partial w_2}$ can be further broken down using the chain rule through the function f_2. Similarly, the gradient with respect to w_1 is:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}, \tag{2.11}$$

where $\frac{\partial \hat{y}}{\partial h_1}$ is taken from the output of function f_2 and $\frac{\partial h_1}{\partial w_1}$ from the output of function f_1. After obtaining the gradients, the weights are updated in the opposite direction of the gradients to minimize the loss, using a *learning rate* α:

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1} \tag{2.12}$$

$$w_2 = w_2 - \alpha \frac{\partial L}{\partial w_2} \tag{2.13}$$

The process is repeated across many iterations, known as epochs, with the forward and backward passes being recalculated each time, gradually improving the model's predictions. The Figure 2.15 illustrates the flow of information during the forward and backward

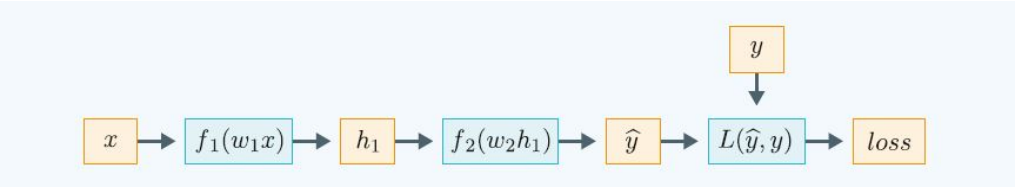

Figure 2.15.: Visualization of a simple neural network model that goes through the backpropagation process [49].

propagation steps.

When building a deep neural network, there are many components that can be used in the model. The ones used in the experiments will be presented here.
Linear is a fundamental component for constructing neural networks, also known as a *fully connected* or *dense layer*. It performs a linear transformation that maps a given set of input data to a new space of output data. This layer applies a weighted sum across input features to produce output features, as shown in equation 2.6 earlier [11].
Dropout is a regularization technique where nodes (from input and hidden layers) along

with their connections are randomly deactivated during training. This process creates a new, thinned network at each training step, which helps to reduce overfitting [3] by preventing complex co-adaptations on training data [5][50][4].
Early Stopping is another regularization technique in neural networks that can be used to stop the training process if there is no improvement after a given number of epochs. It also helps to reduce overfitting, because it does not allow the model to train too much in the training data [6][65].

In the following, the training pipeline is described step by step as it was done in the experiments.

1. Loading and Preprocessing Data: Data is loaded and preprocessed from CSV files, splitting it into features and labels, which are then converted into PyTorch tensors [24].
2. Creating DataLoaders: DataLoaders for training, development, and testing datasets are prepared to manage batch processing and shuffling of data. DataLoaders will be explained more in depth in Chapter 4.
3. Defining the Neural Network with specific layers, activation functions, and output dimensions.
4. Initializing Hyperparameters such as input dimensions, hidden layer size, output dimensions, learning rate, and number of epochs.
5. Instantiate Loss Function and Optimizer [1]: Separate loss functions for different targets (arousal and valence) are defined, and an optimizer is initialized to update network weights based on computed gradients.
6. Training Loop Begins: The model enters a training phase where each epoch consists of:
   - Zeroing Gradients: At the start of each batch, gradients are reset to zero to prevent mixing them up with those from previous batches.
   - Forward Pass: Data is fed forward through the network to compute predictions.
   - Computing Loss: Loss is calculated separately for arousal and valence by comparing predictions with actual labels.
   - Backward Pass: Backpropagation is performed to calculate gradients of the loss with respect to network weights.
   - Update Weights: Weights are updated using the optimizer.
7. Validation Check: At the end of each epoch, the model evaluates performance on the development set to compute metrics like RMSE and MAE for arousal and valence, which will be explained more in depth in Chapter 3.

[3] Overfitting is a common issue in machine learning where a model learns to fit the training data too closely, preventing it from generalizing effectively to new, unseen data. [99]

8. Early Stopping Criteria: The process checks for improvements in validation metrics over a predefined number of epochs, known as patience. If there's no improvement within this period, the best state of the model is saved, and training can be stopped early to prevent overfitting.

9. Saving Model: If the results of the current epoch are better than previous ones, the best metrics are updated and the weights of the model are saved [20].

10. Repeat or Stop: The training continues to the next epoch or it stops if early stopping criteria are met.

2.3.2. Convolutional Neural Networks

Convolutional Neural Network (CNN) is a class of artificial neural networks specifically designed for processing data with a grid pattern, such as images. CNNs are particularly effective in various computer vision tasks due to their ability to automatically and adaptively learn spatial hierarchies of features through a combination of different types of layers: convolutional layers, pooling layers, and fully connected layers [98].

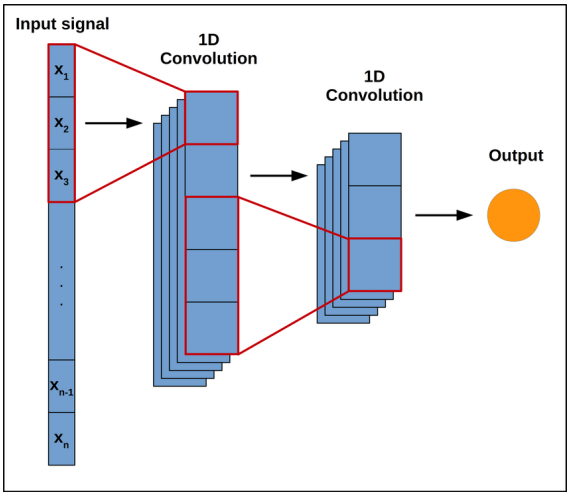

Figure 2.16.: Example of 1D convolutional neural network architecture with two convolutional layers [80].

- Convolutional Layer: The convolutional layer is the fundamental part of the CNN that performs feature extraction, which usually consists of convolution operation and activation function. It applies a set of filters, also known as kernels, to the input image to create feature maps. These feature maps transform the input image into features that become more complex with more layers. Figure 2.16 shows the process of 1D convolution (also known as Conv1D) applied to a signal. It involves sliding a kernel over the input signal, calculating the element-wise product at each position, which extracts features from the signal at various stages and gradually

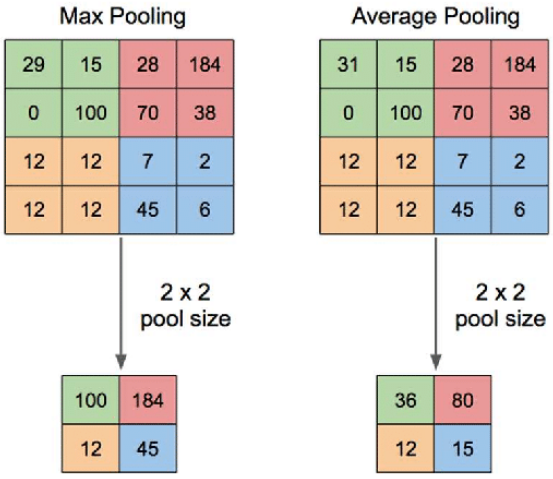

Figure 2.17.: Example of max pooling operation and average pooling with a 2x2 pixel filter size from 4x4 pixel input [97].

transforming the signal through multiple convolution layers until reaching the final output [97][98][80].

- Pooling Layer: Following convolutional layers, pooling layers reduce the in-plane dimensions (width and height, not depth) of the input data for the next convolutional layer. It works by summarizing the features in patches of the feature map into a single value. Max Pooling takes the maximum value from each patch of the feature map. Similarly, Average pooling takes the average value from each patch of the feature map [97][98]. Both are shown in Figure 2.17.

- Fully Connected Layer: These layers are used where neurons are fully connected to all activations in the previous layer. They are typically placed after several convolutional and pooling layers and function as classifiers that use the features from earlier layers. The output can be used to classify the input image into various classes based on the training dataset [98][97].

CNNs have several benefits, including being able to recognize patterns in images regardless of where they appear. This ability, known as shift invariance, means the network can identify the same patterns even if they move around in the image. Additionally, the option to select different kernel sizes allows the network to detect both small, detailed patterns with smaller kernels and larger objects with bigger kernels. In addition, pooling operations not only enhance pattern recognition, particularly for larger objects, but also significantly reduce computational demands and memory usage. This boosts the efficiency and speed of the network, making CNNs effective for complex image analysis tasks [97][49].

In the experiments conducted in Chapter 4, the full CNN architecture is not used since the focus is on regression rather than classification. Instead, some CNN methods are used,

specifically pooling (MAX Pool and AVG Pool) and Conv1D. Temporal pooling is used to compact the information along the sequence data into a single vector representation, therefore making it more manageable [75]. Conv1D is used in the experiments to apply filters on the sequence data, capturing patterns effectively [63].

2.3.3. Recurrent Neural Networks

When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks [**?**]. Recurrent Neural Networks (RNNs) are a class of deep learning models designed to handle sequential data by capturing temporal dependencies. Unlike feed-forward neural networks, RNNs integrate an internal memory that allows them to process inputs considering contextual information. Traditional neural networks don't have the ability to maintain information about previous inputs when processing new data. This limitation is challenging in tasks that involve sequences where the previous data influences the current processing. To address this, RNNs have been developed that allow maintaining a memory of previous computations and therefore enhancing the model's ability to handle sequential data [82]. As shown in Figure 2.18, a

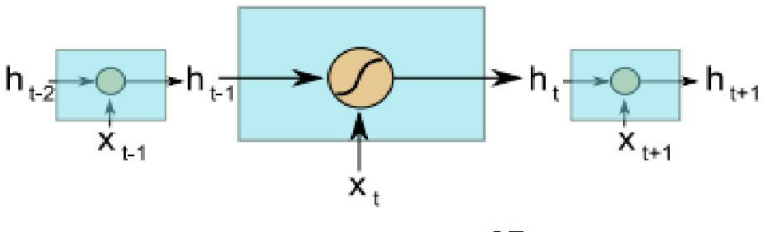

Figure 2.18.: Visualization of a simple RNN internal operation [82].

simple recurrent unit includes a hidden state h_t that is computed based on the current input x_t and the previous hidden state h_{t-1}. This relationship can be captured by the equation:

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{2.14}$$

where h_t is the new hidden state, g is the activation function (typically a hyperbolic tangent), W and U are adjustable weight matrices for the hidden state, b is the bias term, and x denotes the input vector.

Even though simple RNNs excel in sequential data processing, they face limitations in retaining information over long sequences due to their short-term memory. To overcome this, more advanced variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been developed. These architectures introduce mechanisms to selectively retain or forget information, thereby improving the network's ability to learn over longer sequences [82].

The Gated Recurrent Unit (GRU) is a neural network architecture designed to address the limitations of traditional Recurrent Neural Networks (RNNs) ' short-term memory. By simplifying the structure, GRUs offer a more efficient alternative to Long-Short-Term Memory (LSTM) units while maintaining comparable performance. A GRU unit consists of three main components that manage the flow and updating of information within the network: the update gate, reset gate, and current memory content. These components allow the GRU to selectively update and utilize information from previous steps, allowing it to capture long-term dependencies in sequences [82]. The structure of a GRU unit is

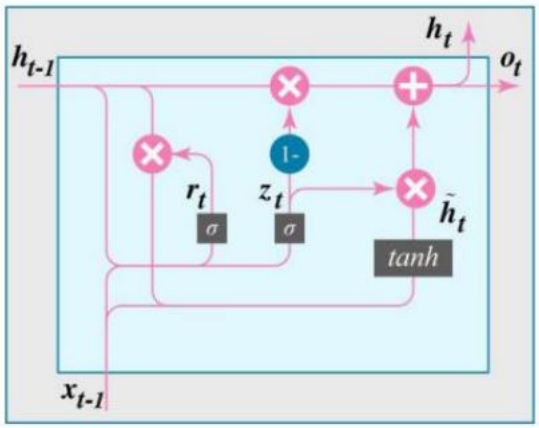

Figure 2.19.: Visualization of a GRU unit [82].

demonstrated in Figure 2.19. The operations are controlled through several gates and states:

- Update Gate (z_t): The update gate decides how much of the past information should be carried forward and is defined by [82]:

  $$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

  It uses a sigmoid function σ to filter the extent to which previous states affect the current state.

- Reset Gate (r_t): The reset gate decides how much of the previous information should be forgotten:

  $$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

  It is computed similarly to the update gate.

- Current Memory Content ($\tilde{h}_t$): It is calculated based on the reset gate and concatenation of the previous hidden state and current input. The result is passed through an activation function:

  $$\tilde{h}_t = \tanh(W_h[r_t \cdot h_{t-1}, x_t])$$

- Final Memory State (h_t): The new states determined by a combination of the previous hidden state and the candidate activation:

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$$

Additionally, an optional output gate (o_t) can be defined to control the information flow from the current memory content to the output:

$$o_t = \sigma(W_o \cdot h_t + b_o)$$

GRUs provide a simpler alternative to LSTMs. They reduce the number of tensor operations required for training, which can lead to faster training times. Since GRU offers a solution for sequence modeling with the ability to capture information across long sequences and has a faster training process, it makes it a good neural network architecture choice for the video model, described later in Chapter 4.

2.3.4. Transformer

In sequence-to-sequence problems, the initial approaches utilized RNNs. However, this architecture faced limitations while working with long sequences because it tended to lose information from initial elements when new ones were added. Vaswani et al. [90] introduced the attention mechanism and transformer model architecture to address this issue.

The Transformer, shown in Figure 2.20, contains an encoder model on the left side and a decoder on the right one. Both have a block of "an attention and a feed-forward network" repeated N times. Let us now address the fundamentals of the transformer architecture.

Self-attention is a process where a sequence of vectors is transformed into another sequence of vectors. Let us say we have input vectors $x_1, x_2, \ldots, x_t$ and aim to produce corresponding output vectors $y_1, y_2, \ldots, y_t$, each with a k-dimension. To generate each output vector y_i, self-attention computes a weighted average of all input vectors. One common method for determining the weights is to use the dot product. The self-attention mechanism has three essential elements: *queries, values, and keys* [40]. In the self-attention process, each input vector x_i is transformed into three roles using weight matrices W_q, W_k, and W_v, representing Query, Key, and Value, respectively. These transformations allow each input vector to interact with itself, making it possible to calculate attention weights and then generate the corresponding output vectors. The query matrix helps determine the output vector y_i, and the key matrix calculates other outputs y_j. The Value matrix finalizes each output after the weights are determined. The weight layers K, Q, and V are applied to the same encoded input. Since each of these matrices comes from the same input, we can apply the attention mechanism of the input vector to itself[66][40].

We use the Q, K, and V matrices to calculate attention scores. For example, in the context of neural machine translations, these scores determine "how much focus to place

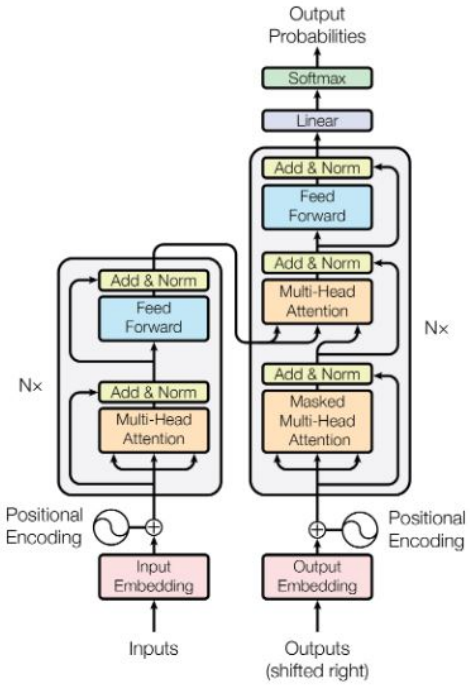

Figure 2.20.: The Transformer architecture [90].

on other places or words of the input sequence w.r.t a word at a certain position." That is, the dot product of the query vector with the key vector of the respective word we're scoring [66]. For example, we calculate the dot products $q_1 \cdot k_1$, $q_1 \cdot k_2$, $q_1 \cdot k_3$, etc. for the first position.
To stabilize gradients, we scale these scores before applying the $softmax$ function. After applying $softmax$, we multiply the results by the Value matrix. The formula for these operations is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.15}$$

Multi-head attention, shown in Figure 2.22, processes the input through multiple attention mechanisms in parallel, allowing it to focus on different parts of the sequence simultaneously(e.g., longer-term dependencies versus shorter-term dependencies). Each independent attention output is computed, concatenated, and linearly transformed into the expected dimension. The weights W are learnable parameters. Typically, this mechanism employs scaled dot-product attention, but it can be swapped out for other types of attention mechanism [90].

Since the Transformer model does not have recurrence and convolution, it needs a way

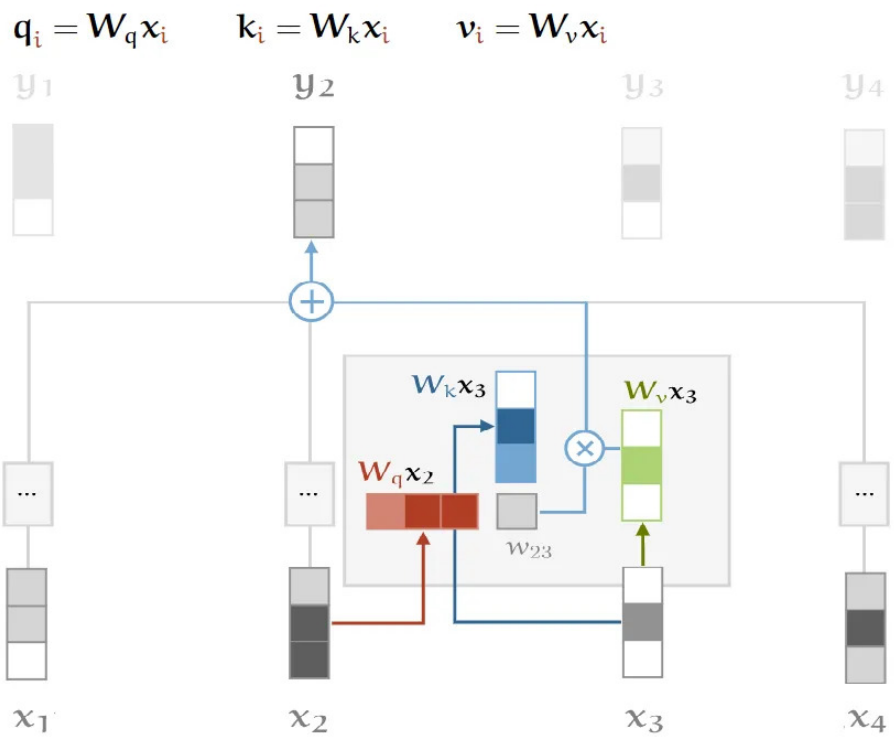

Figure 2.21.: Visualization of the self-attention with key, query, and value transformations [40].

to recognize the sequence order. This is achieved by adding "positional encodings" to the input embeddings at the base of both the encoder and decoder stacks. This provides information about the position of tokens in the sequence.
So, the encoder consists of the following components [90][66]:

- Positional encoding
- Two sublayers: a multi-head self-attention mechanism and a fully connected feed-forward network
- Residual connection around each sublayer for summing up the output
- Layer normalization

The decoder consists of the following components [90][66]:

- Positional encoding

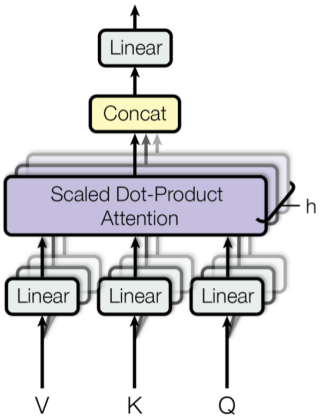

Figure 2.22.: Multi-Head Attention consists of several attention layers running in parallel [90].

- Three sublayers: Masked Multi-head attention to prevent positions from attending to subsequent positions. 'Encoder-decoder attention' (also known as cross-attention), which performs multi-head attention over the output of the decoder. The Key and Value come from the output of the encoder, but the queries come from the previous decoder layer. Afterward, a fully connected network.
- Residual connection around each sublayer for summing up the output
- Layer normalization

In summary, the transformer model consists of two components: an encoder and a decoder. Since transformer can handle long data sequences, outperforms RNN sequence-to-sequence models and provides good results [90], it is the used architecture in the fusion methods experiments.

3. Data and Tools

3.1. AFEW-VA dataset

One of the datasets used in this thesis's experiments is AFEW-VA. It consists of 600 videos extracted from feature films. These videos show various facial expressions captured under challenging indoor and outdoor conditions, including complex, cluttered backgrounds, poor illumination, large out-of-plane head rotations, scale variations, and occlusions. AFEW-VA includes annotations for arousal and valence levels for each frame of the video clips. In total, more than 30,000 frames were annotated with valence and arousal levels in the range of -10 to 10 [59].

As seen in the diagram in Figure 3.1, the distribution of valence and arousal values in the dataset shows a wide range of values, which means there is a variability of emotional expressions in the videos [59]. The distribution of the annotations in the valence and arousal can also be seen on a two-dimensional scatter plot in Figure 3.2.

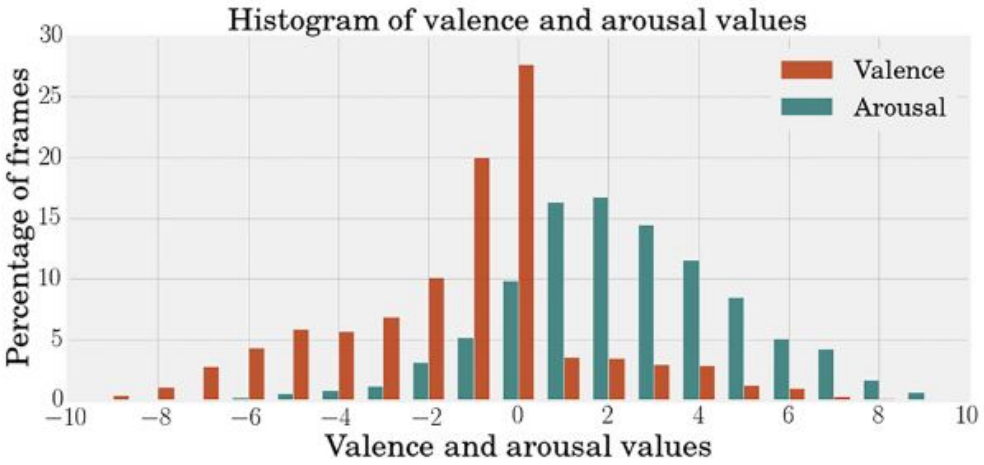

Figure 3.1.: Distribution of arousal and valence values in AFEW-VA dataset [59].

The AFEW-VA dataset addresses the limitations of existing affective datasets by providing detailed annotations for valence and arousal in challenging, real-world conditions in order to understand human emotions in more naturalistic settings [59].

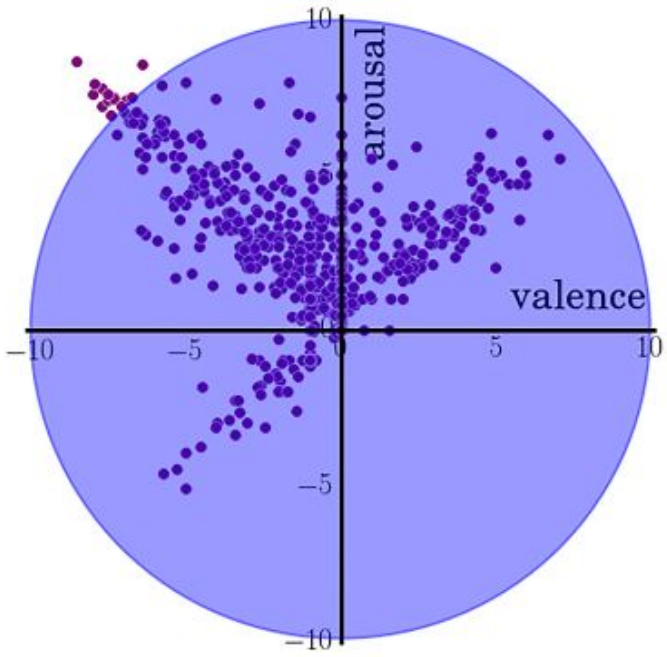

Figure 3.2.: Distribution of valence and arousal values in AFEW-VA, 2D plot [59].

Since the dataset contains a large amount of frames, each of them already annotated with arousal and valence levels, it seems suitable for the experiments in this thesis. AFEW-VA will be used for both the baseline and the video model.

3.2. SEWA dataset

Another dataset that was used in the experiments is SEWA. It consists of audio-visual recordings of spontaneous behavior captured in unconstrained, real-world environments using standard web cameras and microphones. Participants in the dataset were divided into pairs based on their cultural background, age, and gender. They were required to know each other personally to promote natural interactions during the recordings. The people on the recordings are from different age groups, genders, and cultural backgrounds [60]. Kossaifi et al.[60] state that audio-visual data consists of 398 people from six cultures (British, German, Hungarian, Greek, Serbian, and Chinese), 50% female and aged 18 to 65.
The recordings are annotated with facial action units, facial landmarks, vocal and verbal cues, as well as continuously valued emotion dimensions such as valence and arousal, which are crucial for emotion recognition tasks in this thesis.

Figure 3.3 illustrates the distribution of arousal and valence, showcasing the range of emotional variability captured in the recordings. Figure 3.4 displays the same arousal and

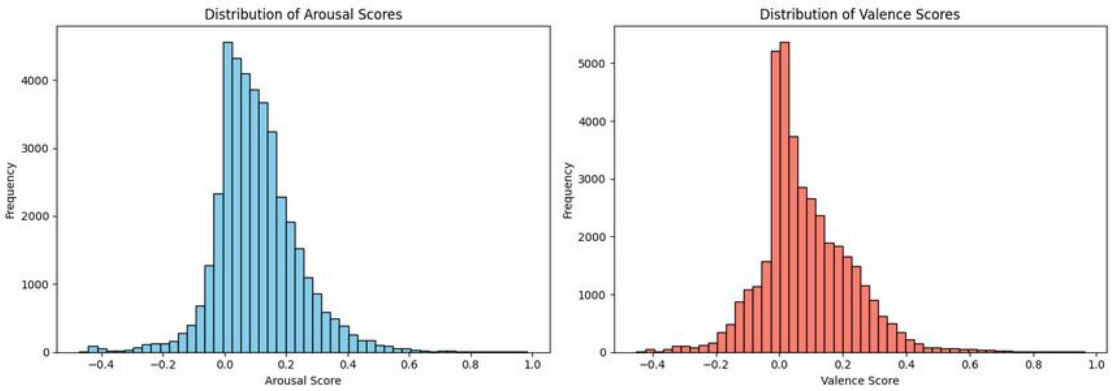

Figure 3.3.: Distribution of arousal and valence values in SEWA dataset.

valence values but on a 2D scatter plot. It is visible that arousal ranges approximately from -0.4 to 1.0, and valence ranges approximately from -0.4 to 0.8.

The SEWA dataset addresses the limitations of existing databases by offering rich annotations in terms of multiple behavioral cues, demographic variability, and task diversity [60].

It seems suitable for emotion recognition task in this thesis too, because it consists of not only visual, but also audio recordings, which will be used in the fusion later on. Also, all the recordings are annotated with arousal and valence values, which are also needed in the experiments. SEWA dataset will be used for the baseline, video and audio model, since it consists of audio recordings, and of course the fusion.

3.3. Feature Extraction

Since the AFEW-VA and SEWA datasets lack the features necessary for predicting arousal and valence, these must be extracted beforehand.

Video feature extraction was executed using an EfficientNet-B1 model, which was pretrained on ten emotion recognition datasets. This initial training gave the model a basic understanding of subtle emotional details in video frames. For each frame, the model extracted a set of 256 deep embeddings. These embeddings were then used as inputs for further processing by emotion recognition models that will be presented in Chapter 4.

Figure 3.5 illustrates a visual representation of the facial feature extraction pipeline. It shows a sequence of processing steps that starts with an input image and then moves through various layers of an EfficientNet-B1 neural network architecture.
Detailed documentation of the model's training process and the datasets incorporated can be accessed on GitHub [9].

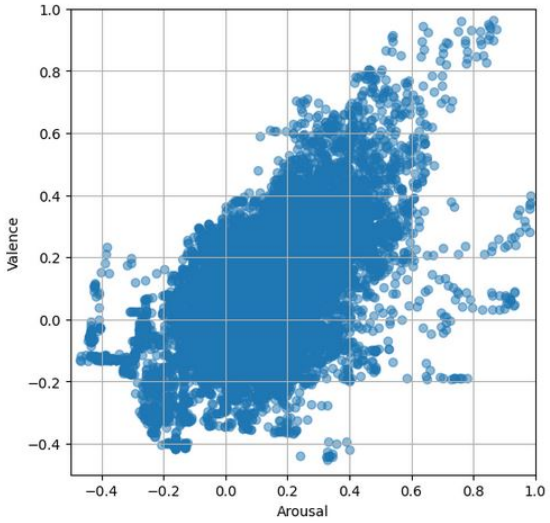

Figure 3.4.: Distribution of valence and arousal values in SEWA, 2D plot.

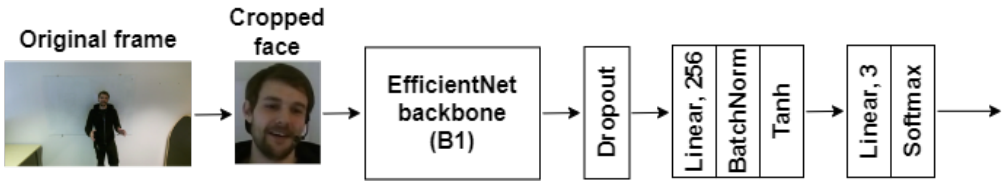

Figure 3.5.: The pipeline of the EfficientNet-B1 model [7].

For the extraction of audio features, the Wav2Vec2 model, which has been pre-trained on 960 hours of Librispeech on 16kHz sampled speech audio, was used [37]. This model features a multilayer convolutional feature encoder, from which we used the last hidden state to extract the audio features. The features were extracted using varying lengths of temporal windows—specifically, 1, 2, 3, and 4 seconds. For each length, the window was moved forward by half its size. More information about the model can be found on the documentation website [1].

Afterward, the features (also called embeddings), along with the file path of each frame and the corresponding arousal and valence values, were stored in a CSV file. In addition, SEWA's video data also included timestamps, and audio data had start and end time steps.

[1] https://huggingface.co/facebook/wav2vec2-base-960h

3.4. Evaluation Metrics

Three metrics, mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE), are used in the experiments to evaluate the performance of regression models. Each provides a different perspective on the error between the predicted and the actual values.

Mean Absolute Error (MAE) is a simple metric that calculates the average of the absolute differences between the predicted and actual values. In other words, it simply says how far away the predictions are from the actual values [87]. It is calculated using the following formula.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|\,, \tag{3.1}$$

where n is the number of predictions, y_i are the actual observed values, and $\hat{y}_i$ are the predicted values.

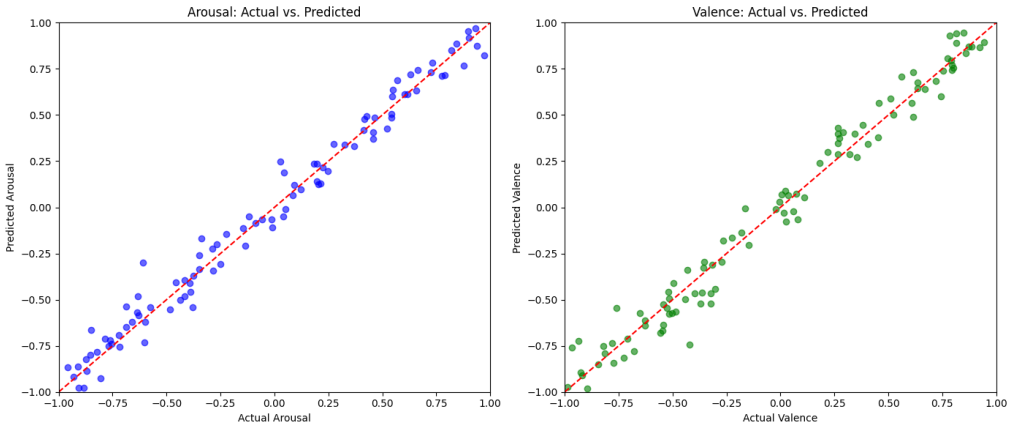

Figure 3.6.: Visual representation of MAE using dummy data, where arousal MAE = 0.08 and valence MAE = 0.10.

Figure 3.6 presents two plots for a visual understanding of MAE. They show the actual and predicted values for arousal and valence. The red dashed line represents where points would lie if the predictions were correct. The closer the points are to the dashed line, the better the model's predictions.
Since this metric is easily interpretable from the raw value alone, it is useful for evaluating neural networks because it deals with many different calculations that need to be compared in the experiments. The use of absolute values of the differences also makes it more robust to outliers [12].

Mean Squared Error (MSE) is the average of the squared difference between predicted and actual values, so it is always a positive value. This squaring of the differences has

the effect of amplifying larger errors. In other words, big mistakes are even bigger when they are squared, which makes them stand out more. This helps the model correct these mistakes quickly during its learning process. Also, when the mistakes are small and close to each other, the model learns faster [87][32].
MSE has the advantage of being a continuous and differentiable function, which means it is well-suited for gradient-based optimization algorithms [12]. For this reason, it is used in experiments with machine learning algorithms like SVR and Bayesian Ridge. Even though the Decision tree algorithm is not gradient-based, using MSE as a metric allows a consistent comparison between models. The MAE metrics is determined by the following formula.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.2}$$

where n is the number of predictions, y_i are the actual observed values, and $\hat{y}_i$ are the predicted values.

Root Mean Squared Error (RMSE) is the square root of MSE, which brings the error measurement back to the original units of the output variable. RMSE works by squaring the differences, summing them, dividing them by the number of observations, and finally taking the square root, which gives the standard deviation of the prediction errors. This indicates how close the line of best fit is to the set of points [87].
It is similar to MAE, but RMSE is more sensitive to outliers because it gives more weight to larger errors, whereas MAE treats all errors equally [21]. RMSE can be calculated with this following formula.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.3}$$

In summary, we use the combination of MSE and RMSE in experiments with algorithms and combination of MAE and RMSE in neural networks to capture advantages of all metrics in used models. In all three metrics, a smaller value represents that the predictions are closer to the actual values, which means the prediction performance is better, so the goal is to find the best model with the smallest results during experiments.

4. Experiments

This chapter contains all the experiments on the SEWA and AFEW-VA datasets. Execution of the code was done only on one system, which is powered by an AMD Ryzen 9 5950X with 16 cores. The system has an NVIDIA GeForce RTX 3060 graphics card and 131 GB of memory. Python served as the programming language, and two packages were used during the experiments: PyTorch [18] for the development of neural networks and scikit-learn [22], which was used for implementing machine learning algorithms such as Support Vector Regression (SVR), Bayesian Ridge (BR), Decision Tree Regression (DTR), as well as regression evaluation metrics.

4.1. Data Manipulation

4.1.1. Data Loading

Before starting with the experiments, specific data manipulation is necessary. Firstly, the dataset had to be split into training, development (also known as validation), and test subsets. It is a standard practice in machine learning. The reason behind this is simple: the model needs to have a separate train dataset to learn patterns from the data and relationships between features and outputs. The development dataset is used to tune the model's hyperparameters, allowing developers to adjust the model for better performance. Test data is used to evaluate the model's performance on completely unseen data [25].

The split ratio used in the experiments is 70% train data, 15% development data, and 15% test data. It is important to note that the split between train/dev/test was the same across all experiments. Since we have separate audio and video datasets when performing fusion, it was necessary to ensure that the models are evaluated under the same conditions [23]. So, in the experiments, one of the steps in the beginning is always loading the data. This means that all three subsets, train, dev, and test, were loaded.

4.1.2. Data Preprocessing

In the experiments, there are three ways how data is preprocessed, depending on the model. They all integrate the embeddings' extraction described in Chapter 3.

For the baseline neural network model, after extracting the embeddings and saving them in a CSV file, each frame that consists of embeddings (features and labels) is transformed into a 1D tensor. Afterward, each frame with embeddings is put into a 2D tensor with the size (m,n), where m are frames and n are features with labels. It is then passed to

a DataLoader. The DataLoader is used in every model in the experiments since it can effectively manage the data during the training process. It groups data into manageable sizes (creates batches), performs shuffling on the data to ensure data randomness, and handles parallel processing to make data loading quicker [19]. So, DataLoader outputs batches of data, which are then fed into the input layer of a neural network model. The visual representation of the data preprocessing for the baseline model can be seen in Figure A.6.

For the video model, each frame is transformed into a 1D tensor and sequences of frames (windows) are created using a window size that was initialized at the beginning. Then, we need to iterate through each row in the data frame and match the frames with the corresponding embeddings. Then, the embeddings are converted into a PyTorch tensor. Similarly, that window's average arousal and valence labels are also converted into a PyTorch tensor. DataLoader then fetches batches of windowed frame data and corresponding labels during model training, validation, or testing. Figure A.7 presents the data preprocessing process for the video model.

In the fusion model, the preprocessing is more complex because it includes synchronization of video and audio data based on timestamps. First, video IDs from file paths in the video data frame are extracted. This ID is used to group video and audio data by video, ensuring synchronization within each video separately. We filter out the corresponding video and audio data for each video ID. This is done by selecting rows from the video data frame where the path contains the video ID and from the audio data frame where the filename contains the video ID. We experiment with different window lengths in each fusion, so this should be considered while synchronizing video and audio, too. So, we ensure that the timestamps align in video and audio. Afterward, the last video frame's arousal and valence scores are used as labels for the window. Features are also extracted from video and audio data within the window. The synchronized data is then stored in a tuple consisting of the video features, audio features, labels, video name, and timestamp. Figure A.8 shows the simplified data preprocessing workflow for fusion models.

4.2. Baseline

4.2.1. Machine Learning Algorithms

Before the investigation of fusion methods and applying fusion started, we needed to build a simple model first- a baseline. It served as a starting point from which we could compare how well the complex fusion methods models perform [2].
In this section, the results of three machine learning algorithms are presented: Support Vector Regression, Bayesian Ridge, and Decision Tree Regression. The goal is for the results of the baseline deep learning neural network, which will be discussed next, to match with or have slightly better results than those of the algorithms. Let us discuss the results of the AFEW-VA dataset first.

In Bayesian Ridge Regression model, first, datasets were loaded from CSV files and preprocessed by extracting features x and target values y. Since the range of arousal and valence values in AFEW-VA dataset is -10 to 10, additional normalization is needed. It is achieved by simply dividing the values by 10. Separate regression models were trained for arousal and valence using the training data.
Similarly, in Decision Tree Regression, datasets were loaded, features extracted, and target variables normalized by division by 10. It is also important to note that DTRs were initialized with a fixed random state, which controls the randomness involved during the node-splitting process and, therefore, ensures reproducibility [27]. Afterward, the training on the arousal and valence data took place.
Lastly, in Support Vector Regression, in addition to loading datasets and extracting features, a grid search was conducted to optimize the SVR parameters (including parameter C and kernel type) using a predefined split [17] that combined training and development data for cross-validation, ensuring model validation before testing. The best models for valence and arousal were then used to predict arousal and valence on development and test data. The performance of all three algorithms was evaluated using the development and test data using MSE and RMSE metrics. Table 4.1 presents the results of the described

Table 4.1.: Results of the used machine learning algorithms on AFEW-VA. In the following, in each table v = valence, a = arousal

Algorithm	MSE (V)	RMSE (V)	MSE (A)	RMSE (A)
Bayesian Ridge	0.009	0.09	0.019	0.13
SVR	0.009	0.09	0.017	0.13
Decision Tree	0.023	0.15	0.039	0.19

algorithms in the AFEW-VA dataset. Bayesian Ridge and SVR got approximately the same result. In both, the RMSE for valence equals 0.09, which means that the model's predictions are generally close to the actual data points, with an average error of 0.09 units. RMSE for arousal equals 0.13, meaning that the average magnitude of the errors in the predictions of arousal by the model is 0.13 units. Valence prediction appears to be more accurate than arousal prediction. The reason for this can be the differences in distribution patterns between arousal and valence in the AFEW-VA dataset. When looking back at Figure 3.1 in Chapter 3, it is visible that the valence values are more distributed in the center with a peak around zero. So, extreme positive and negative values are less common.
On the other hand, arousal values have a more even distribution, from negative to positive. Unlike valence, there is no single dominant category. Therefore, it makes it more challenging to predict the arousal values accurately. Also, by looking at Figure 3.2, it can be seen that valence is more concentrated around the center, while arousal is spread along the vertical axis.
This means that the worse arousal results can be due to the increased variability and spread in arousal values, which makes it harder for the model to predict the outcomes

accurately. On the contrary, the concentration of valence values around the center is beneficial to the model since there is less extreme variation.
Unlike BR and SVR, the Decision Tree Regression algorithm ended up having worse results. RMSE for arousal is 0.19, and RMSE for valence is 0.15. This can be due to the fact that DTR is prone to overfitting, meaning it predicts well for the training data but can be inaccurate for new data [16]. As established earlier in Chapter 2, BRR and SVR are more regularized forms of algorithms compared to DTR. Bayesian Ridge naturally prevents overfitting by including prior distributions on the model parameters and therefore avoids too complex models. SVR can handle non-linear data better by using kernel tricks and C parameter.

Now, let us discuss the SEWA results. The algorithms' implementation stays the same, except for the differences in the datasets. As seen in Table 4.2, in all three algorithms, the

Table 4.2.: Results of the used machine learning algorithms on SEWA.

Algorithm	MSE (V)	RMSE (V)	MSE (A)	RMSE (A)
Bayesian Ridge	0.01	0.10	0.01	0.10
SVR	0.01	0.11	0.01	0.10
Decision Tree	0.03	0.17	0.02	0.14

results of valence prediction appear to be slightly worse than arousal. In contrast to the AFEW-VA dataset, in SEWA, the distribution of arousal is more evened out. However, valence seems to be leaning towards positive values with a peak around 0.2 to 0.4, as seen in Figure 3.3. Positive valence values are more common in the dataset than neutral or negative values.
The distribution of valence can still be considered normal, except for the concentration of data points around the specific range. So, the slightly worse valence results can be due to the fact that models may struggle to accurately predict less frequent, extreme values. By looking at Figure 3.4, we can see a concentration around the center but also a broad spread along both axes, meaning a wide range of values in the data.
Here, too, the Decision Tree algorithm produces worse results than the other two algorithms. This suggests that Decision Tree Regression is not the best model choice for SEWA and AFEW-VA datasets in this emotion recognition task.

4.2.2. Neural Network

Now, let us look at the neural network's baseline results. The model's structure looks alike for both AFEW-VA and SEWA datasets.
It consists of:

1. Data loading and data preprocessing, which are described in Section 4.1.
2. Feed Forward Neural Network with a simple architecture, which can be seen in Figure 4.1 with:

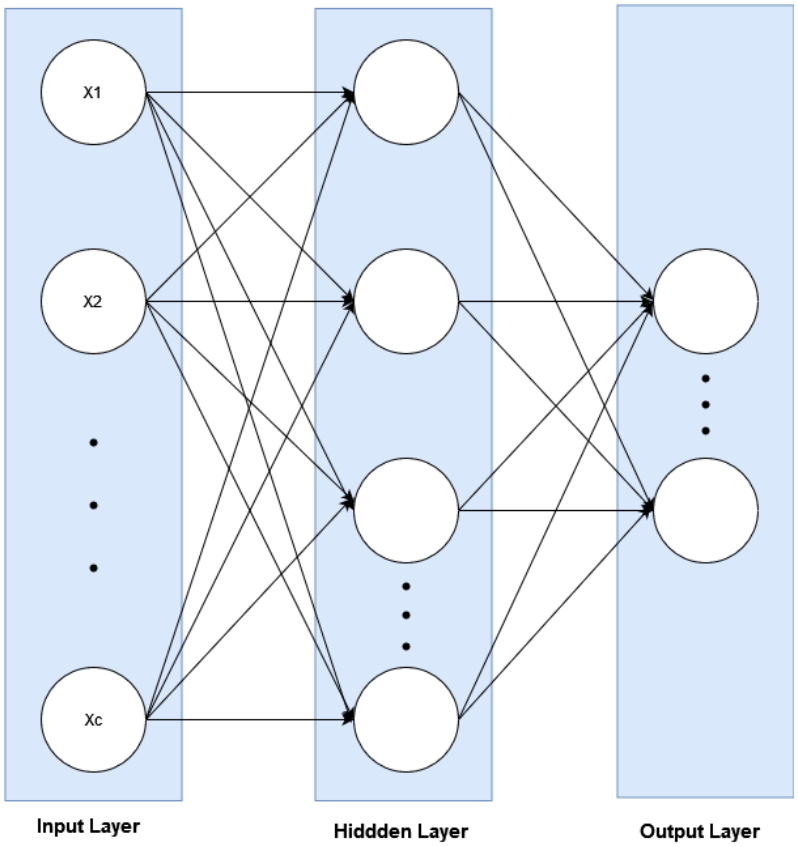

Figure 4.1.: Visual representation of the Feed Forward Neural Network used in the baseline experiments. The variables x in the input layer are batches that are processed sequentially one after the other.

- Input Layer
- Hidden Layer followed by ReLU activation function
- Output Layer followed by Tanh activation for scaling the output

3. Training setup with Loss Function (RMSE), optimizer and early stopping.
4. Training loop with epochs, batches, loss calculation and validation.
5. Evaluation with MAE and RMSE metrics.

Baseline on AFEW-VA	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
Feed Forward Neural Network	0.07	0.09	0.08	0.13

Table 4.3.: Results for the baseline Feed Forward Neural Network on AFEW-VA

Baseline with SEWA	MAE (V)	RMSE (V)	MAE (A)	RMSE (A)
Feed Forward Neural Network	0.09	0.12	0.08	0.10

Table 4.4.: Results for the baseline Feed Forward Neural Network on SEWA

When comparing the results of the FFNNs from table 4.3 and table 4.4 with those from the algorithms, we can see that the RMSE results are mostly the same. This means that all the models, except the Decision Tree, predict arousal and valence values well. The MAE results in both datasets suggest that the neural networks perform well. In SEWA, an MAE of 0.09 for valence means that, on average, the model's predictions are off by 0.09 units from the actual values. MAE of 0.10 for arousal shows that the model's predictions are off by 0.10 units on average. Similarly, in the AFEW-VA dataset, the predictions for valence are off by 0.07 units and for arousal by 0.13 units on average. Considering that the range of arousal and valence is frothe m -1 tof o 1, the error rate seems low.

It is crucial that the frame-to-frame FFNNs provide good results since the following more complex experiments build upon the baseline.

4.3. Video Model

Next, we discuss the experiments on the video model. Since the model has to process several frames at a time and make predictions while considering them as one video, the model changes from frame-to-frame to sequence-to-one. Here, the Feed-Forward Neural Network no longer works, so we have to use another type of neural network—a Recurrent Neural Network, specifically GRU. GRU performs well when processing sequence data for predictions, where the context from previous inputs is important in order to understand current inputs.

Figure 4.2 shows the architecture of the used video model, which consists of the following:

1. Input Layer
2. GRU Layer, which can handle sequences of data
3. Dropout Layer, which randomly deactivates some nodes with their connections to prevent overfitting
4. Pooling: Temporal pooling (1D pooling) is applied to reduce the sequence length and summarize the features across the time dimension. This makes the data simpler by either averaging out (AVG Pool) the features over the sequence or selecting the maximal value from the features over the sequence (MAX Pool). `Conv1d` is used to extract features from the sequence data by applying filters to the sequence to capture patterns over the sequence that may be important for the performance of the following layers.

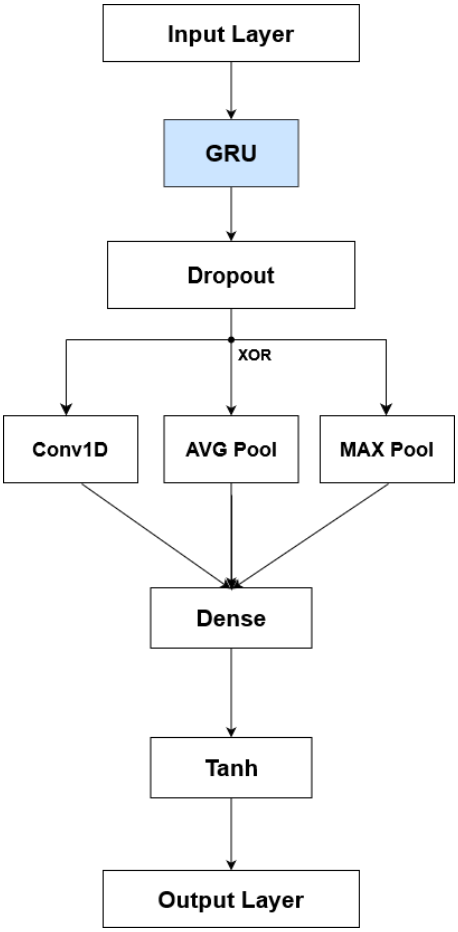

Figure 4.2.: Visual representation of the RNN-based video model used in the experiments.

5. Dense (Fully Connected) Layer, which maps the pooled features to the output size of arousal and valence
6. Tanh activation function, which is used to transform the output to be in the range -1 to 1
7. Output Layer

Now let us take a look at the results of the video models. Overall, 12 experiments were conducted on each dataset. For each window size (1, 2, 3 or 4 seconds) one of the three pooling types was applied. We will focus on the best results instead of displaying all results.

The best results of the AFEW-VA video model can be seen in Table 4.5.
Interestingly, the MAX Pool has the best result on the AFEW-VA dataset for every amount of seconds (1 second is not included in the best results, but it still beats the other

Seconds, window size	Pooling	MAE (V)	RMSE (V)	MAE (A)	RMSE (A)
2 sec, 10	MAX Pool	0.07	0.11	0.06	0.07
3 sec, 15	MAX Pool	0.07	0.11	0.06	0.08
4 sec, 20	MAX Pool	0.07	0.10	0.06	0.08

Table 4.5.: Best results of the video-based model on the AFEW-VA dataset

pooling approaches as seen in Table A.1). In addition, a bigger window size brings the best results. That can be because everything longer than 1 second has a longer context. Compared to the baseline, this video model has the same result in MAE for valence but better in MAE for arousal. Even though the baseline model predicted arousal values worse than valence, in this video model, the prediction performance of arousal improved with more contextual data. All results of the AFEW-VA video model can be found in Table A.1.

Seconds, window size	Pooling	MAE (V)	RMSE (V)	MAE (A)	RMSE (A)
2 sec, 10	AVG Pool	0.08	0.10	0.08	0.11
3 sec, 15	AVG Pool	0.07	0.10	0.08	0.11
4 sec, 20	AVG Pool	0.08	0.10	0.08	0.11

Table 4.6.: Best results of the video-based model on the SEWA dataset

Table 4.6 shows the best results of the video model on the SEWA dataset. Here, AVG Pool worked out best on all window sizes (again, 1 second is not included in the best results, but it beats the other pooling approaches by looking at Table A.2). The results with 2, 3, and 4 seconds all seem similar, but 3 seconds appear to be the best among them. It has the same MAE for arousal but a slightly better MAE for valence. It appears that 3 seconds is an optimal value since it is not too long, like 4 seconds, but also not too short, like 1 or 2 seconds. Compared to the baseline, MAE for arousal stayed the same, but MAE for valence improved. So, even though the valence prediction performance was worse in the baseline model, in this video model it improved with more contextual data, too, like in AFEW-VA. All results of the SEWA video model can be found in Table A.2.

Conv1D approach had bad results compared to AVG Pool and MAX Pool in both datasets. The exact reason behind this is unknown, but it can be because the convolutional filters are not effectively capturing the relevant features, or arousal and valence may have specifics in their temporal dynamics that are better captured by summarizing information over a sequence (with AVG and MAX Pools) instead of trying to detect specific patterns using

a convolutional approach. Either way, Conv1D seems to be not the best approach choice in this video model for the AFEW-VA and SEWA datasets in this emotion recognition task.

4.4. Audio Model

Before discussing the fusion results, we need to examine the audio model's results because fusion is based on audio and video. Four experiments were conducted using the audio model on only the SEWA dataset since AFEW-VA contains no audio.

As stated earlier in Section 3.3, four audio CSV files contain extracted features from 1,2,3 and 4-second temporal windows. The architecture stayed the same as in the baseline SEWA model described earlier, which can also be seen in Figure 4.1.

Seconds	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
1 sec	0.13	0.16	0.13	0.17
2 sec	0.13	0.16	0.13	0.18
3 sec	0.13	0.16	0.13	0.18
4 sec	0.13	0.16	0.13	0.18

Table 4.7.: Performance of the audio-based model on the SEWA dataset

The results of the audio model can be seen in Table 4.7. All of them are the same, so there is no visible observation regarding temporal window length, as observed in the video model. Overall, the audio model performed worse than the video model. It can be because audio often consists of more noise. A person can be, e.g., silent on the video, so the model has nothing to predict. A face is generally more expressive than audio, which gives better results in prediction performance in the SEWA video model.

Even though the AFEW-VA dataset contains no audio, the video model was still built using this dataset because the video model is more significant for emotion recognition. Therefore, it made sense to look at the prediction performance of both AFEW-VA and SEWA to be able to compare them.

4.5. Audio-Visual Fusion

Lastly, this section presents the results of various fusion methods. Overall, four fusion methods were selected. Each of them was tested on a different window size, just like the video and audio models. Additionally, the last fusion method had three variations. So, in total, 24 experiments were conducted.
Let us go through the methods, starting from the simplest one. The architecture of fusion

1, shown in Figure 4.3 consists of the following:

1. Data loading and data preprocessing (synchronization of video and audio)
2. AVG Pool: Video/audio features for each window are pooled across time dimension, reducing them to a single vector
3. Concatenation of pooled video and audio features
4. Dense Layer (Fully Connected Layer with ReLU activation function)
5. Output Layer

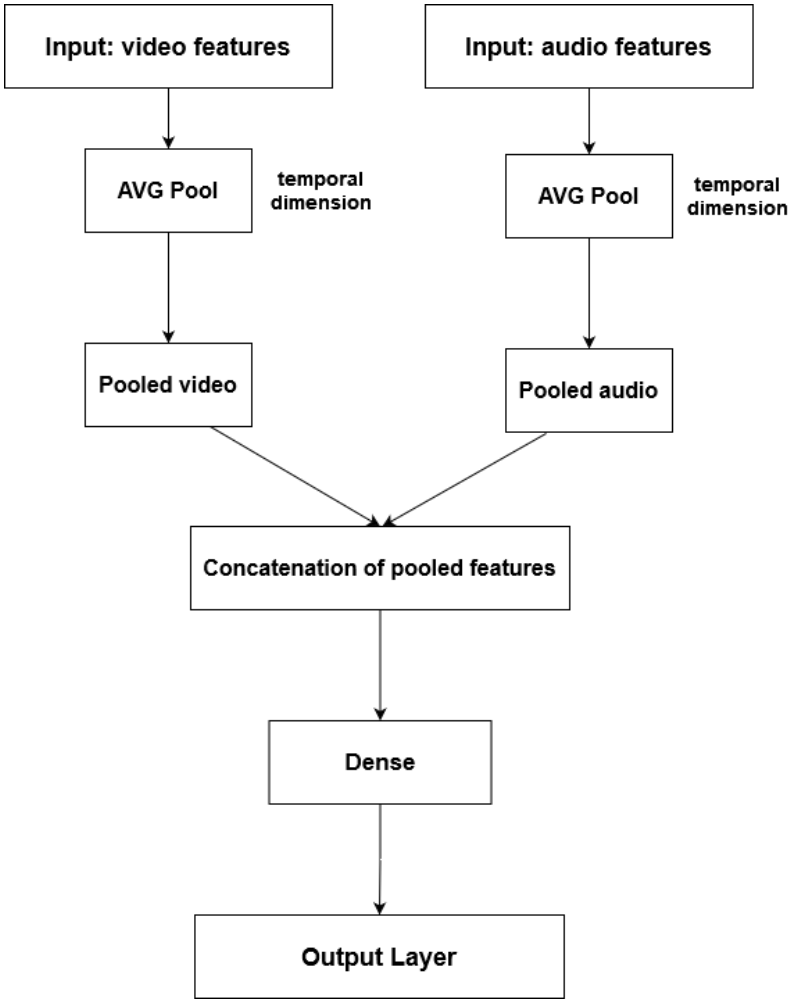

Figure 4.3.: Visual representation of fusion method 1.

Starting from Fusion 2, the methods get more complex. They now utilize the modern deep learning architecture - a transformer.
Fusion 2, as shown in Figure A.1 consists of:

1. Data loading and data preprocessing (synchronization of video and audio)
2. Linear Layer, where audio features are normalized to match the dimension of the video features
3. Concatenation of normalized audio features and video features
4. 2 Transformer Blocks (self-attention mechanisms) which have:
   - Multi-Head-Attention
   - Add and Norm
   - Position-wise Feed-Forward Network, where a fully connected feed-forward network is applied separately to each position
   - Positional Encoding, which provides information about each element's position in the sequence. It is important because it helps the model understand the sequence order
5. AVG Pool, which is applied to convert the sequence of output features from the Transformer into a single feature vector
6. Output Layer

Fusion 3 is similar to Fusion 2, but it also incorporates cross-attention. It, as shown in Figure A.2 consists of:

1. Data loading and data preprocessing (synchronization of video and audio)
2. Linear Layer and Padding, where audio features are normalized to match the dimension of the video features. Afterward, the audio features were padded to match the number of timesteps with those in the video
3. Transformer block (cross-attention mechanism), which processes the video and padded audio features together
4. Transformer block (self-attention mechanism), which further processes the output of the cross-attention layer
5. AVG Pool
6. Output Layer

Lastly, fusion 4 has three different variations, but they all incorporate only cross-attention. Here, as shown in Figure A.3, is the architecture of the first variation of fusion 4:

1. Data loading and data preprocessing (synchronization of video and audio)
2. Linear Layer, where audio features are normalized to match the dimension of the video features
3. Conv1D Layer, where dimensions of video features are adjusted
4. Transformer block (cross-attention mechanism), which processes the encoded video features as query and the encoded audio features as key and value.
5. Transformer block (self-attention mechanism), which processes the output of the cross-attention layer
6. AVG Pool
7. Linear and Output Layer, where the pooled features are passed through a fully connected layer to predict arousal and valence

The second variation, as shown in Figure A.4, has the following architecture:

1. Data loading and data preprocessing (synchronization of video and audio)
2. Linear Layer, where audio features are normalized to match the dimension of the video features
3. Conv1D Layer, where dimensions of video features are adjusted
4. Transformer block (cross attention mechanism), which processes the encoded video features and the encoded audio features.
5. Transformer block (cross attention mechanism), which reapplies cross-attention to the output of the first cross-attention layer, which is integrated as query and video features are integrated as keys and values
6. AVG Pool
7. Linear and Output Layer, where the pooled features are passed through a fully connected layer to predict arousal and valence

The third variation is almost identical to the second variation, except that in the second transformer block, audio features are integrated as keys and values instead of video. The architecture of the third variation is shown in Figure A.5.

Table 4.8 demonstrates the best result of each fusion method. It seems that each fusion type has better results on different window length. In Fusion 2 and 4 (variation 2), 4 seconds length has the best results in MAE for valence and MAE for arousal. This means that these types of fusion benefit from longer contextual information and therefore predict outcomes better. Fusion 1 and 3, on the other hand, work better on shorter window

	Seconds, window size	**Pooling**	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
Fusion 1	2 sec, 10	AVG Pool 2x	0.0893	0.1145	0.0930	0.1183
Fusion 2	4 sec, 20	AVG Pool	0.0902	0.1174	0.0988	0.1284
Fusion 3	1 sec, 5	AVG Pool	0.0928	0.1206	0.1025	0.1355
Fusion 4(v2)	4 sec, 20	1D CNN, AVG Pool	0.0815	0.1035	0.0956	0.1239

Table 4.8.: Results of the best fusion method on test data (SEWA dataset).

	Seconds, window size	**Pooling**	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
Fusion 1	2 sec, 10	AVG Pool 2x	0.0606	0.0830	0.0689	0.0972
Fusion 2	4 sec, 20	AVG Pool	0.0601	0.0892	0.0639	0.0904
Fusion 3	1 sec, 5	AVG Pool	0.0606	0.0822	0.0709	0.0985
Fusion 4(v2)	4 sec, 20	1D CNN, AVG Pool	0.0583	0.0809	0.0679	0.0940

Table 4.9.: Results of the best fusion methods on development data.

length and have the best performance on 1 and 3 seconds, respectively.

If we compare arousal and valence prediction performance separately, it can be observed that with longer window length, meaning longer contextual information, valence prediction gets better in Fusion 4. In contrary, arousal prediction are worse with longer window length. One possible reason is that the audio data, after synchronizing it with video, "confuses" the model and makes the predictions worse. The best audio-based model had a window length of 1 second, where arousal predictions were slightly better than valence. So, it makes sense that the fusion model performs arousal predictions better on shorter window lengths.

In addition, it is observable that Fusion 4 (variation 2) performed better than Fusion 4 (variation 3), as seen in Table A.3. The first one had video features integrated into the second cross-attention mechanism and the other audio features. So this also leads to the conclusion that by incorporating additional audio features, they drag the overall performance of the fusion model down, even though the video model alone used to perform better. Similarly, the Fusion 4 (v2) model is better at predicting valence than baseline, but is slightly worse in predicting arousal than the baseline.

When we compare the results on test data with the results on demonstration data in Table 4.9, we see that the model learned way better on demonstration data but has worse

results on test data. This means that the fusion models adapt worse to new, unseen data. If we now look back at the video model, it has better results than the fusion methods, which means that the video model's generalization is better than the generalization of the fusion models.

Overall, after carrying out all the fusion experiments, we can say that the audio-visual fusion models outperform the audio model. This is probably due to the fact that the video model alone performs well at prediction arousal and valence values. The video modality alone still has better results than the fusion. However, these experiments provide promising results regarding future work in the multimodal emotion regonition task, showing that even on challenging data, the fusion either has the same results as a model based on one individual modality alone or even outperforms it.

5. Conclusion

This work explores various audio-visual fusion methods in the context of emotion recognition. Before proceeding with fusion approaches, video, and audio models were built first. The video model provided good results, which were better than the baseline model results. The audio model's results were worse than the video, presumably due to the noise in the audio data.

For the fusion approaches, the data was preprocessed first, which involved synchronizing audio and video using a customized PyTorch DataLoader. In the experiments, four different fusion approaches were investigated. One was a simple method incorporating a fully connected layer, and the other three incorporated the modern deep learning architecture - a transformer. The best result of each fusion method was evaluated, and it showed that the best results of all four fusion approaches outperformed the audio model, and two of them had the same result as the video model in valence prediction. The arousal prediction ended up being slightly worse because the audio data seemed to drag the overall performance of the fusion methods down.

In future work, we can experiment with different databases using the same fusion approaches and analyze how well the fusion model performed compared to baseline models. The hyperparameters of the models also could be adjusted, like early stopping or a number of epochs, to see if that provides better results. Another possibility could be to try incorporating different architectures into the fusion models. The work of Huang et al. [51] shows that combining transformer with LSTM further improves the performance and achieves good results. It can be experimented whether Transformer-LSTM architecture has a better prediction performance.

This thesis shows that audio-visual fusion achieves better results than audio modality alone and has as good results as valence in video modality, which shows the potential for further improving fusion methods to achieve better results.

A. Appendix

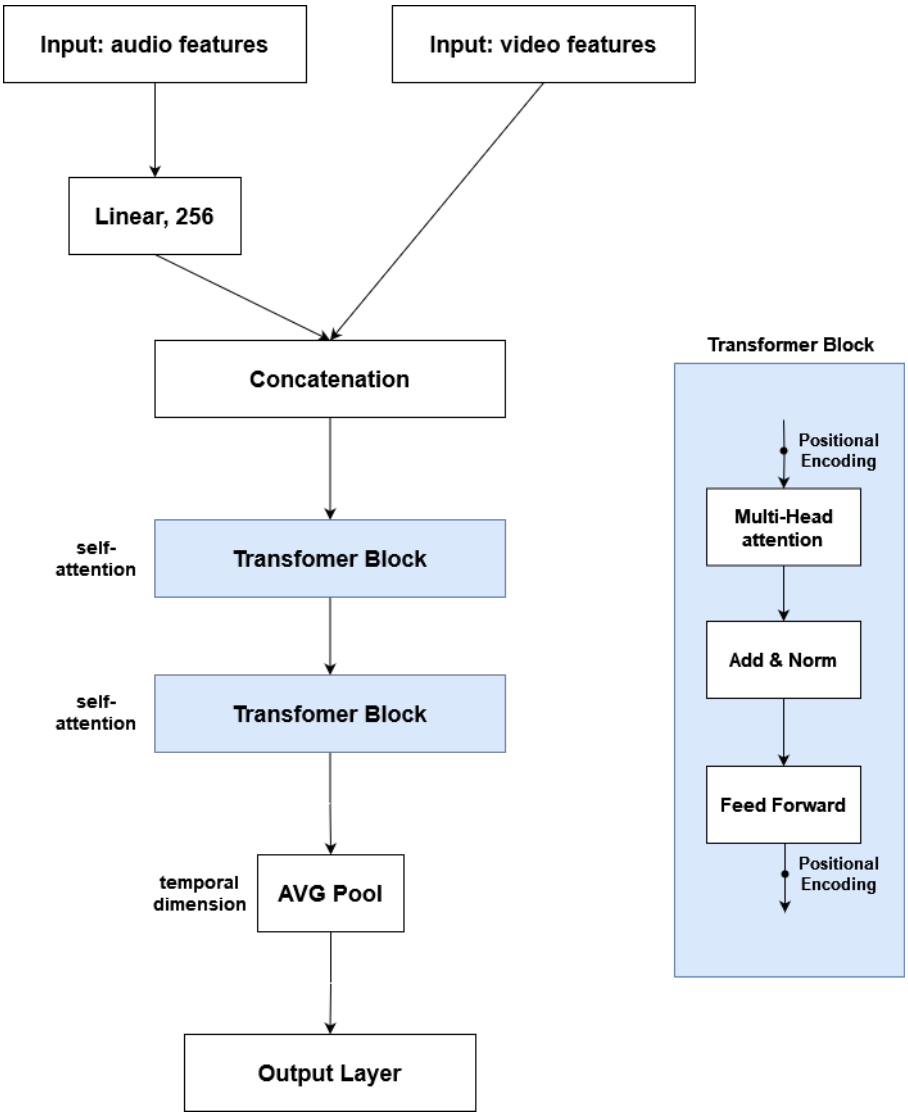

Figure A.1.: Visual representation of fusion method 2.

Table A.1.: Video-based model on AFEW-VA dataset

Seconds, window size	**Pooling**	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
1 sec, 5	AVG Pool	0.09	0.12	0.06	0.08
	MAX Pool	0.08	0.11	0.07	0.09
	1D CNN	0.77	0.82	1.19	1.22
2 sec, 10	AVG Pool	0.09	0.13	0.07	0.09
	MAX Pool	0.07	0.11	0.06	0.07
	1D CNN	0.77	0.82	1.20	1.23
3 sec, 15	AVG Pool	0.08	0.12	0.07	0.09
	MAX Pool	0.07	0.11	0.06	0.08
	1D CNN	0.77	0.82	0.79	0.84
4 sec, 20	AVG Pool	0.09	0.12	0.06	0.08
	MAX Pool	0.10	0.10	0.07	0.08
	1D CNN	1.23	1.26	0.79	0.83

Table A.2.: Video-based model on SEWA dataset

Seconds, window size	**Pooling**	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
1 sec, 5	AVG Pool	0.08	0.11	0.09	0.12
	MAX Pool	0.09	0.12	0.09	0.12
	1D CNN	0.95	0.96	0.95	0.96
2 sec, 10	AVG Pool	0.08	0.10	0.08	0.11
	MAX Pool	0.08	0.11	0.09	0.11
	1D CNN	1.04	1.05	0.95	0.96
3 sec, 15	AVG Pool	0.07	0.10	0.08	0.11
	MAX Pool	0.08	0.11	0.08	0.11
	1D CNN	1.04	1.05	1.04	1.05
4 sec, 20	AVG Pool	0.08	0.10	0.08	0.11
	MAX Pool	0.09	0.12	0.08	0.11
	1D CNN	0.95	0.96	0.95	0.96

Table A.3.: Fusion methods on SEWA dataset (test data)

Method name	**Seconds, window size**	**Pooling**	**MAE (V)**	**RMSE (V)**	**MAE (A)**	**RMSE (A)**
Fusion 1	1 sec, 5	AVG Pool 2x	0.0876	0.1113	0.0944	0.1213
Fusion 1	2 sec, 10	AVG Pool 2x	0.0893	0.1145	0.0930	0.1183
Fusion 1	3 sec, 15	AVG Pool 2x	0.0923	0.1174	0.0975	0.1246
Fusion 1	4 sec, 20	AVG Pool 2x	0.0941	0.1197	0.0999	0.1297
Fusion 2	1 sec, 5	AVG Pool	0.0900	0.1232	0.1021	0.1289
Fusion 2	2 sec, 10	AVG Pool	0.0923	0.1201	0.1033	0.1327
Fusion 2	3 sec, 15	AVG Pool	0.0914	0.1193	0.1013	0.1292
Fusion 2	4 sec, 20	AVG Pool	0.0902	0.1174	0.0988	0.1284
Fusion 3	1 sec, 5	AVG Pool	0.0928	0.1206	0.1025	0.1355
Fusion 3	2 sec, 10	AVG Pool	0.1058	0.1401	0.1034	0.1328
Fusion 3	3 sec, 15	AVG Pool	0.0925	0.1203	0.1092	0.1470
Fusion 3	4 sec, 20	AVG Pool	0.1129	0.1494	0.1057	0.1350
Fusion 4 (v1)	1 sec, 5	1D CNN, AVG Pool	0.1371	0.1835	0.1336	0.1682
Fusion 4 (v1)	2 sec, 10	1D CNN, AVG Pool	0.1151	0.1533	0.1064	0.1346
Fusion 4 (v1)	3 sec, 15	1D CNN, AVG Pool	0.1165	0.1548	0.1097	0.1391
Fusion 4 (v1)	4 sec, 20	1D CNN, AVG Pool	0.1196	0.1589	0.1042	0.1346
Fusion 4 (v2)	1 sec, 5	1D CNN, AVG Pool	0.0861	0.1106	0.0932	0.1208
Fusion 4 (v2)	2 sec, 10	1D CNN, AVG Pool	0.0867	0.1098	0.0937	0.1210
Fusion 4 (v2)	3 sec, 15	1D CNN, AVG Pool	0.0827	0.1050	0.0976	0.1261
Fusion 4 (v2)	4 sec, 20	1D CNN, AVG Pool	0.0815	0.1035	0.0956	0.1239
Fusion 4 (v3)	1 sec, 5	1D CNN, AVG Pool	0.1379	0.1852	0.1337	0.1683
Fusion 4 (v3)	2 sec, 10	1D CNN, AVG Pool	0.1385	0.1855	0.1341	0.1678
Fusion 4 (v3)	3 sec, 15	1D CNN, AVG Pool	0.1379	0.1842	0.1374	0.1713
Fusion 4 (v3)	4 sec, 20	1D CNN, AVG Pool	0.1377	0.1836	0.1368	0.1719

Method name	Seconds, window size	Pooling	MAE (V)	RMSE (V)	MAE (A)	RMSE (A)
Fusion 1	1 sec, 5	AVG Pool 2x	0.0697	0.0811	0.0689	0.0958
Fusion 1	2 sec, 10	AVG Pool 2x	0.0606	0.0830	0.0689	0.0972
Fusion 1	3 sec, 15	AVG Pool 2x	0.0626	0.0857	0.0718	0.1009
Fusion 1	4 sec, 20	AVG Pool 2x	0.0642	0.0888	0.0718	0.1010
Fusion 2	1 sec, 5	AVG Pool	0.0694	0.0814	0.0666	0.0943
Fusion 2	2 sec, 10	AVG Pool	0.0602	0.0820	0.0667	0.0940
Fusion 2	3 sec, 15	AVG Pool	0.0602	0.0816	0.0661	0.0924
Fusion 2	4 sec, 20	AVG Pool	0.0601	0.0892	0.0639	0.0904
Fusion 3	1 sec, 5	AVG Pool	0.0606	0.0822	0.0709	0.0985
Fusion 3	2 sec, 10	AVG Pool	0.0619	0.0848	0.0727	0.1012
Fusion 3	3 sec, 15	AVG Pool	0.0627	0.0848	0.0712	0.0992
Fusion 3	4 sec, 20	AVG Pool	0.0641	0.0877	0.0715	0.1009
Fusion 4 (v1)	1 sec, 5	1D CNN, AVG Pool	0.0900	0.1163	0.1026	0.1341
Fusion 4 (v1)	2 sec, 10	1D CNN, AVG Pool	0.0735	0.0988	0.0735	0.1037
Fusion 4 (v1)	3 sec, 15	1D CNN, AVG Pool	0.0666	0.0881	0.0740	0.1028
Fusion 4 (v1)	4 sec, 20	1D CNN, AVG Pool	0.0661	0.0877	0.0717	0.0992
Fusion 4 (v2)	1 sec, 5	1D CNN, AVG Pool	0.0588	0.0813	0.0696	0.0987
Fusion 4 (v2)	2 sec, 10	1D CNN, AVG Pool	0.0600	0.0826	0.0683	0.0984
Fusion 4 (v2)	3 sec, 15	1D CNN, AVG Pool	0.0594	0.0822	0.0679	0.0944
Fusion 4 (v2)	4 sec, 20	1D CNN, AVG Pool	0.0583	0.0809	0.0672	0.0940
Fusion 4 (v3)	1 sec, 5	1D CNN, AVG Pool	0.0906	0.1166	0.1023	0.1337
Fusion 4 (v3)	2 sec, 10	1D CNN, AVG Pool	0.0910	0.1171	0.1001	0.1326
Fusion 4 (v3)	3 sec, 15	1D CNN, AVG Pool	0.0922	0.1195	0.1022	0.1341
Fusion 4 (v3)	4 sec, 20	1D CNN, AVG Pool	0.0908	0.1193	0.1034	0.1363

Table A.4.: Fusion methods on SEWA dataset (development data)

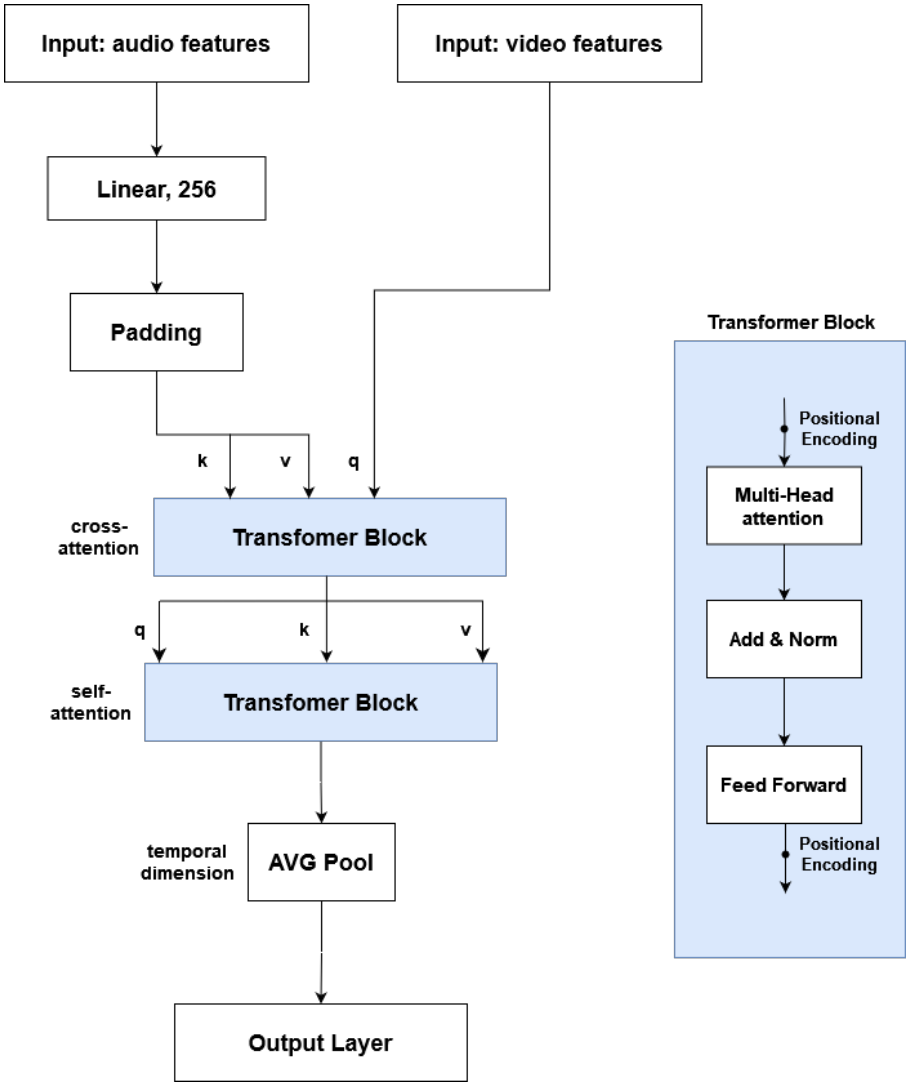

Figure A.2.: Visual representation of fusion method 3.

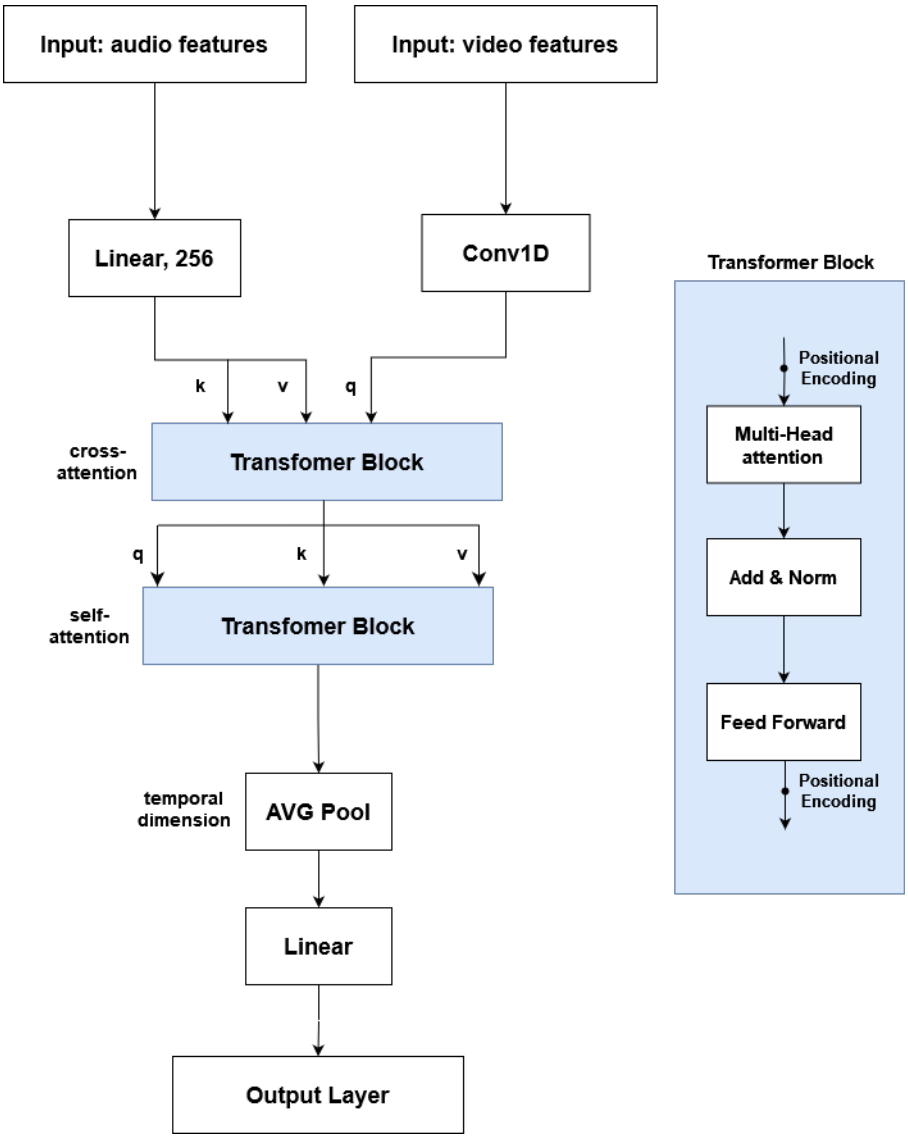

Figure A.3.: Visual representation of fusion method 4, variation 1.

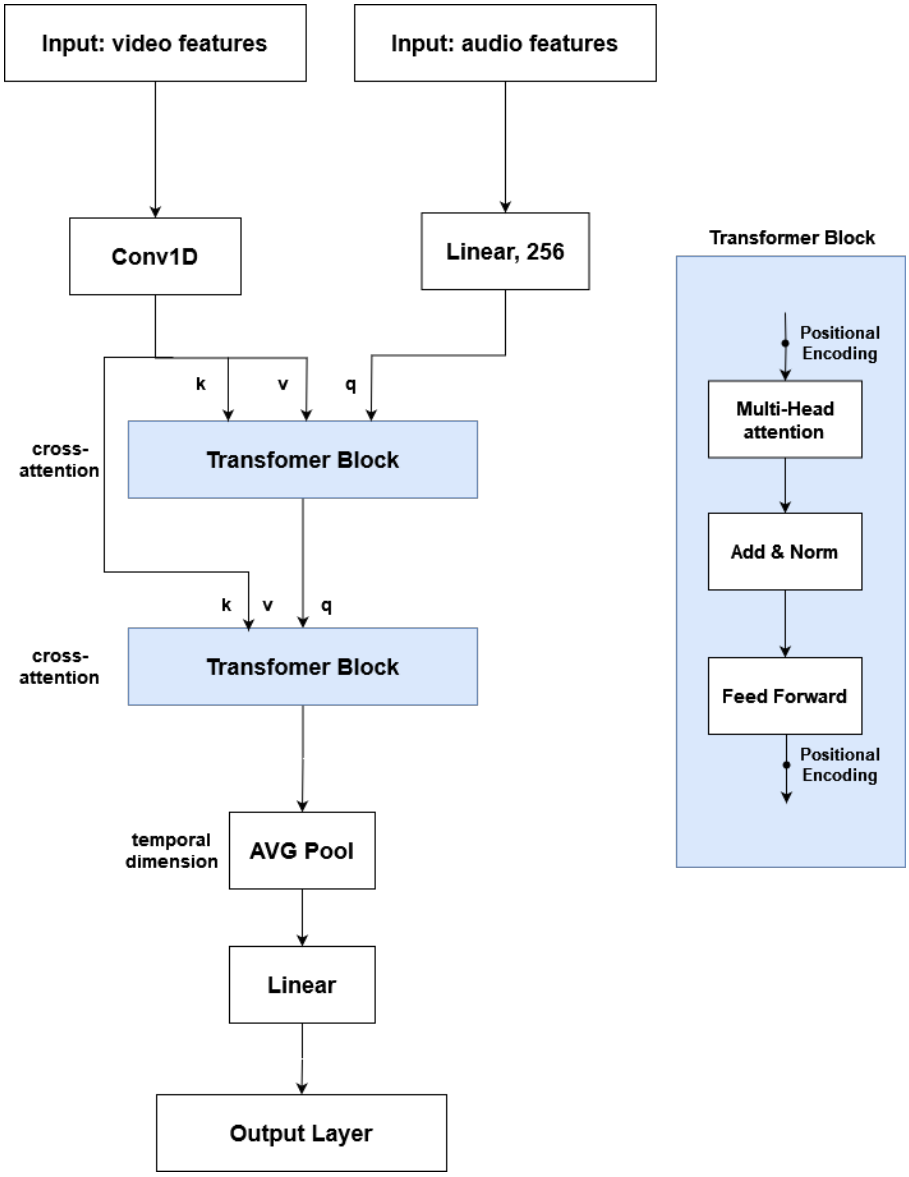

Figure A.4.: Visual representation of fusion method 4, variation 2.

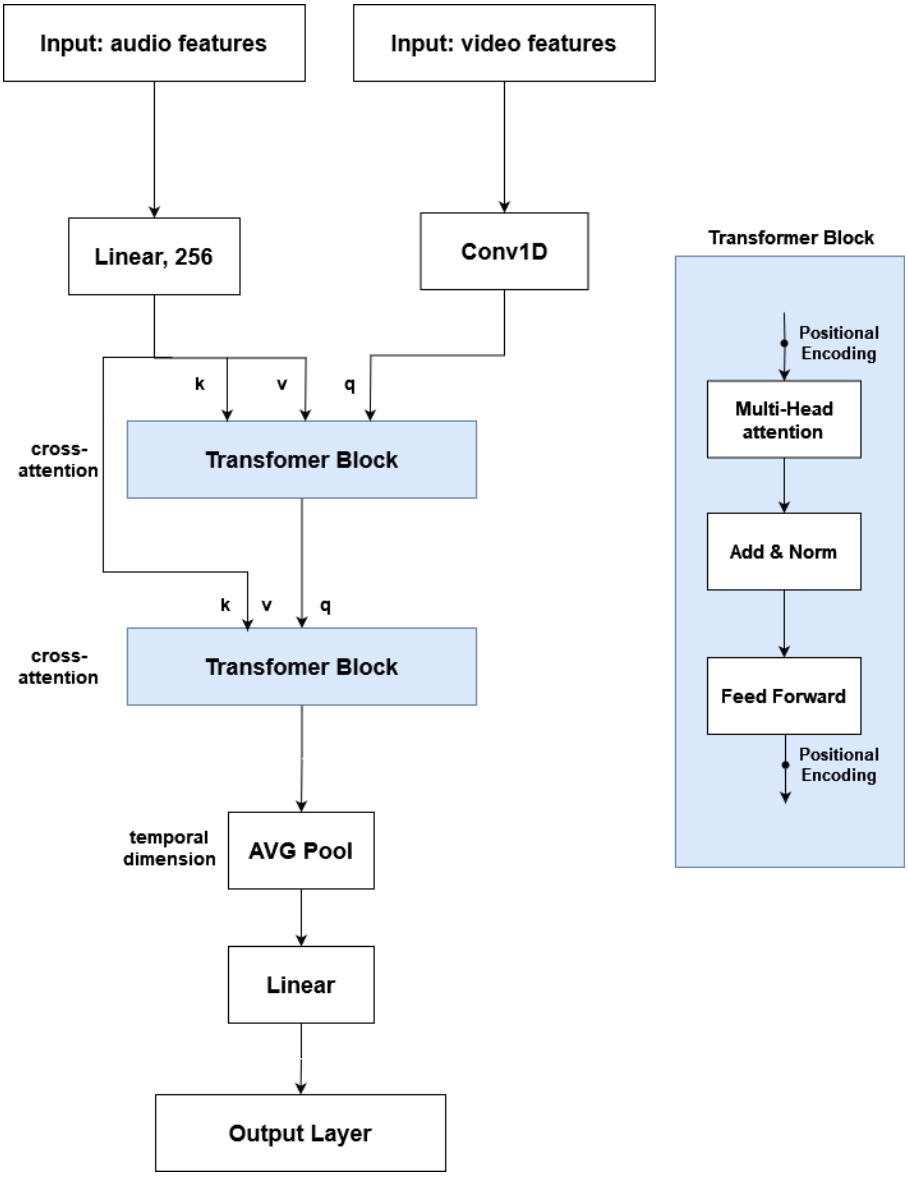

Figure A.5.: Visual representation of fusion method 4, variation 3.

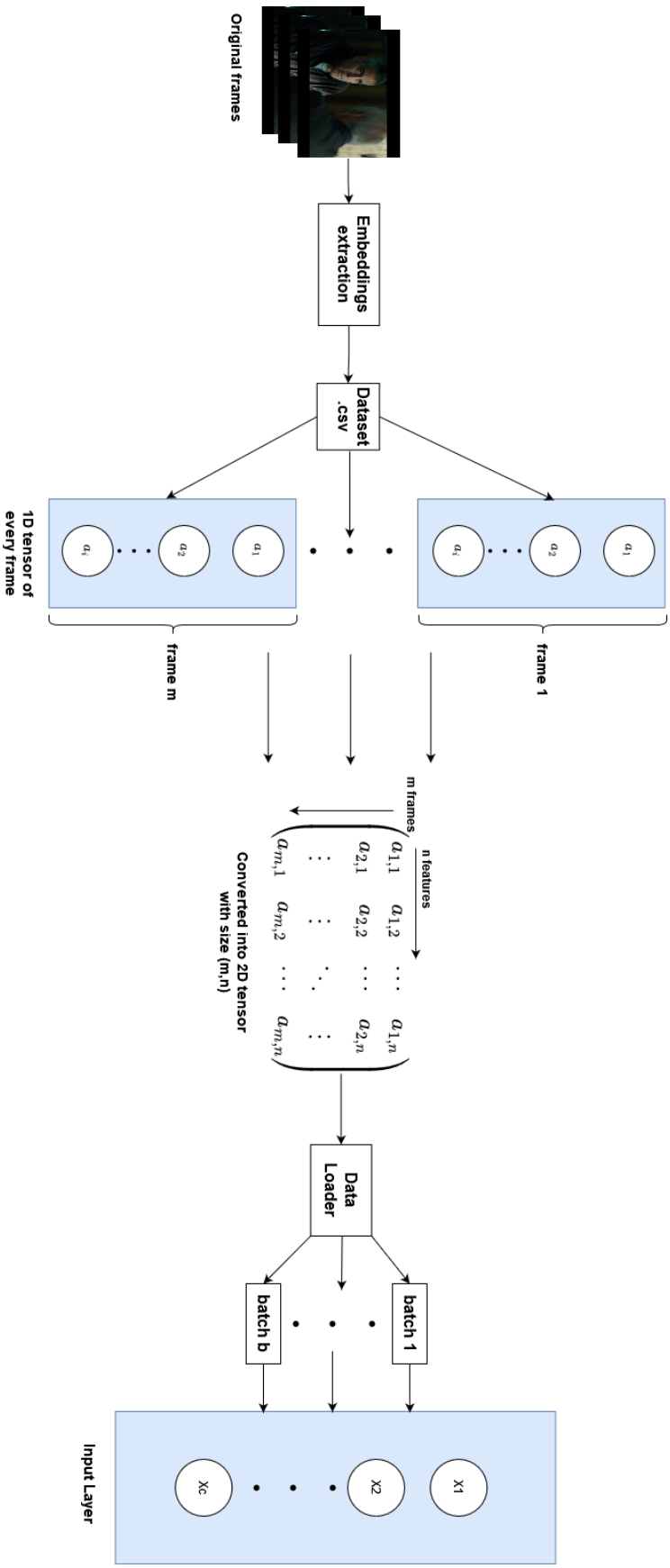

Figure A.6.: Data preprocessing in the baseline model.

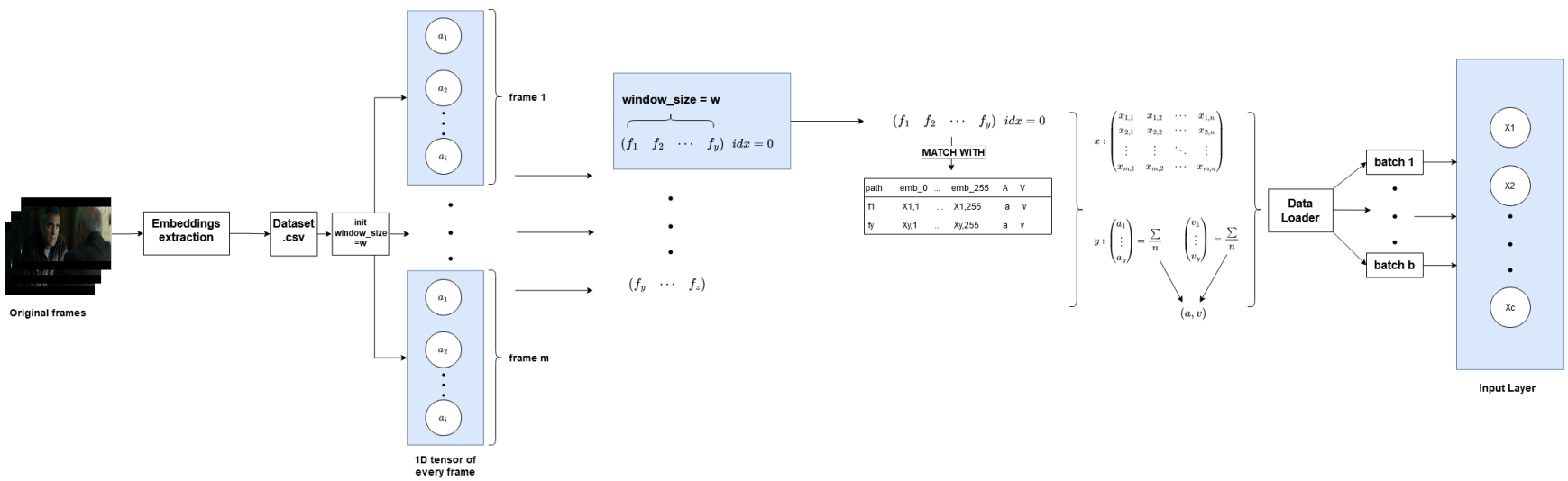

Figure A.7.: Data preprocessing in the video model.

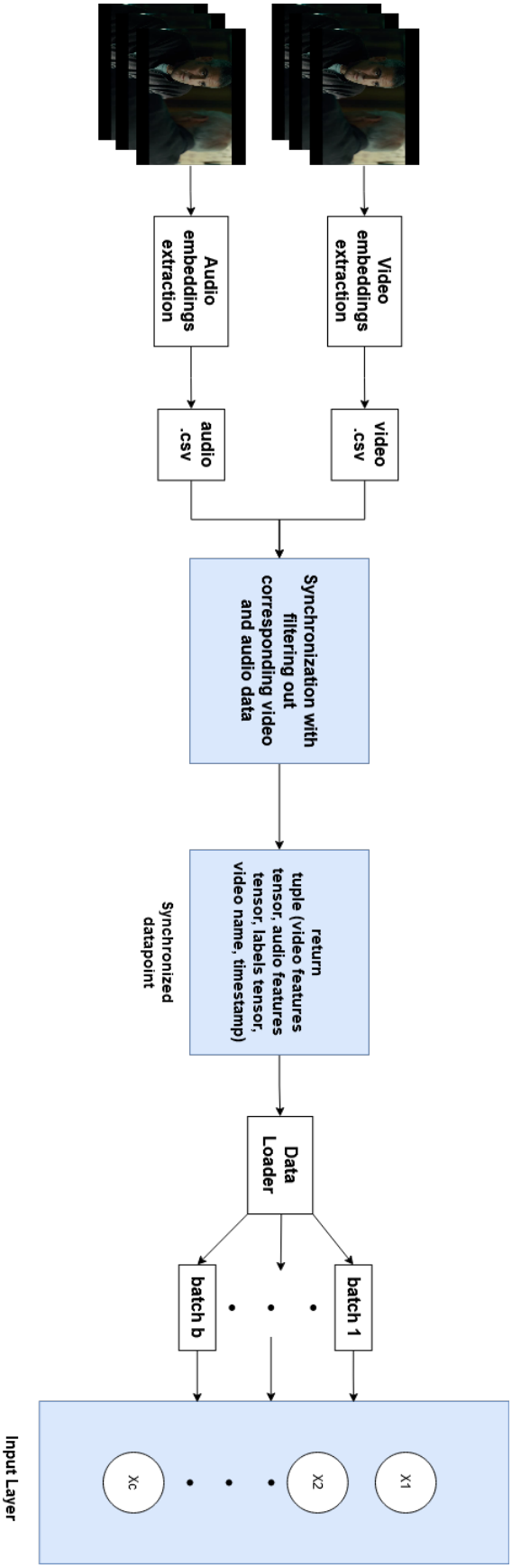

Figure A.8.: Data preprocessing in the audio-visual model.

Bibliography

[1] Adam optimization algorithm, pytorch documentation. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html [Accessed: 2024-04-18].

[2] Baselines. https://medium.com/@mohsenmostafa833/baselines-3f702e143e49 [Accessed: 2024-04-18].

[3] Deep learning 101: Beginners guide to neural network. https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/ [Accessed: 2024-04-18].

[4] Dropout in neural networks. https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9 [Accessed: 2024-04-18].

[5] Dropout pytorch class. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html [Accessed: 2024-04-18].

[6] Early stopping pytorch class. https://pytorch.org/ignite/generated/ignite.handlers.early_stopping.EarlyStopping.html [Accessed: 2024-04-18].

[7] Efficientnet-b1 model for facial feature extraction. https://github.com/ECCV-2024-cross-multi-modal/Cross-Multi-Modal-Fusion-Approach/tree/main/emotion_recognition [Accessed: 2024-04-18].

[8] Fundamentals of deep learning – activation functions and when to use them? https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ [Accessed: 2024-04-18].

[9] Github documentation of video features extraction process. https://github.com/ECCV-2024-cross-multi-modal/Cross-Multi-Modal-Fusion-Approach/tree/main/emotion_recognition [Accessed: 2024-04-18].

[10] Linear models. https://scikit-learn.org/1.0/modules/linear_model.html#ridge-regression [Accessed: 2024-04-18].

[11] Linear pytorch class. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html [Accessed: 2024-04-18].

[12] Loss functions unraveled. https://medium.com/@ompramod9921/loss-functions-unraveled-805d76c239fe [Accessed: 2024-04-18].

[13] Multicollinearity. https://en.wikipedia.org/wiki/Multicollinearity [Accessed: 2024-04-18].

[14] Ordinary least squares regression (ols). https://www.xlstat.com/de/loesungen/eigenschaften/ordinary-least-squares-regression-ols [Accessed: 2024-04-18].

[15] Outlier. https://en.wikipedia.org/wiki/Outlier [Accessed: 2024-04-18].

[16] Overfitting and pruning in decision trees — improving model's accuracy. https://medium.com/nerd-for-tech/overfitting-and-pruning-in-decision-trees-improving-models-accuracy-fdbe9ecd1160 [Accessed: 2024-04-18].

[17] Predefinedsplit. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.PredefinedSplit.html [Accessed: 2024-04-18].

[18] Pytorch. https://pytorch.org/ [Accessed: 2024-04-18].

[19] Pytorch dataloader documentation. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html [Accessed: 2024-04-18].

[20] Pytorch documentation on saving and loading models. https://pytorch.org/tutorials/beginner/saving_loading_models.html [Accessed: 2024-04-18].

[21] Rmse and mae. https://medium.com/@vaibhav1403/rmse-and-mae-415470f52b58 [Accessed: 2024-04-18].

[22] scikit-learn. https://scikit-learn.org/stable/ [Accessed: 2024-04-18].

[23] Splitting into train, dev and test sets; stanford lecture. https://cs230.stanford.edu/blog/split/ [Accessed: 2024-04-18].

[24] Tensor pytorch documentation. https://pytorch.org/docs/stable/tensors.html [Accessed: 2024-04-18].

[25] Training, validation, test split for machine learning datasets. https://encord.com/blog/train-val-test-split/ [Accessed: 2024-04-18].

[26] Understanding hidden layers in neural networks. https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning [Accessed: 2024-04-18].

[27] Why do we set a random state in machine learning models? https://towardsdatascience.com/why-do-we-set-a-random-state-in-machine-learning-models-bb2dc68d8431 [Accessed: 2024-04-18].

[28] "lasso & ridge regression" in 200 words. `https://thaddeus-segura.com/lasso-ridge/` [Accessed: 2024-04-18].

[29] *Affect and Emotion in Human-Computer Interaction: From Theory to Applications*. Springer Berlin Heidelberg, 2008.

[30] Support vector machines, 2024. `https://scikit-learn.org/stable/modules/svm.html` [Accessed: 2024-04-18].

[31] Tuning the hyper-parameters of an estimator, 2024. `https://scikit-learn.org/stable/modules/grid_search.html` [Accessed: 2024-04-18].

[32] Nirajan Acharya. Choosing between mean squared error (mse) and mean absolute error (mae) in regression. `https://medium.com/@nirajan.acharya666/choosing-between-mean-squared-error-mse-and-mean-absolute-error-mae-in-regression-a-deep-dive-c16b4eeee603` [Accessed: 2024-04-18].

[33] Beny Maulana Achsan. Support vector machine: Regression, 2019. `https://medium.com/it-paragon/support-vector-machine-regression-cf65348b6345` [Accessed: 2024-04-18].

[34] Charu C Aggarwal. *Neural networks and deep learning*. Springer International Publishing, Cham, Switzerland, 1 edition, September 2018.

[35] Laith Alzubaidi, Jinglan Zhang, Humaidi Amjad J., Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaria, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Springer Link*, 2021.

[36] A. George Assaf, Mile Tsionas, and Anastasios Tasipoulos. Diagnosing and correcting the effects of multicollinearity: Bayesian implications of ridge regression. *Elsevier*, pages 1–2, 2018.

[37] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv*, 2020.

[38] D. Basak, S. Pal, and D. C. Patranabis. Support Vector Regression. *Semantic Scholar*, page 203, 2007.

[39] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Springer Link*, page 1040, 2017.

[40] Peter Bloem. Transformers from scratch.

[41] Francesca M.M Citron, Marcus A. Gray, Hugo D. Critchley, Weekes Brendan S., and Evelyn C. Ferstl. Emotional valence and arousal affect reading in an interactive way: Neuroimaging evidence for an approach-withdrawal framework. *Elsevier*, pages 79,81, 2014.

[42] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2012.

[43] Paul Ekman. An argument for basic emotions. *Cognition and Emotion*, 6(3–4):169–200, May 1992.

[44] Ana Garcia-Acosta, Jorge De la Riva, Jaime Sanchez, and Rosa Maria Reyes-Martinez. Neuroergonomic Stress Assessment with Two Different Methodologies, in a Manual Repetitive Task-Product Assembly. *ResearchGate*, 2021.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] Pooja Gulati, Amita Sharma, and Manish Gupta. Theoretical Study of Decision Tree Algorithms to Identify Pivotal Factors for Performance Improvement: A Review. *ResearchGate*, 2016.

[47] Teddy Surya Gunawan, Muhammad Nurrudin Muktaruddin, Mira Kartiwi, Yasser Asrul Ahman, and Tina Dewi Rosahdi. Development of Video-Based Emotion Recognition System using Transfer Learning. *IEEE*, pages 1,5, 2022.

[48] Syrine Haddad, Olfa Daassi, and Safya Belghith. Emotion Recognition from Audio-Visual Information based on Convolutional Neural Network. *IEEE*, page 2, 2023.

[49] Dirk Hecker and Gerhard Paass. *Künstliche Intelligenz. Was steckt hinter der Technologie der Zukunft?* Springer Link, 2020.

[50] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*, 2012.

[51] Jian Huang, Jianhua Tao, Bin Liu, Zheng Lian, and Mingyue Niu. Multimodal Transformer Fusion for Continuous Emotion Recognition. *IEEE*, pages 2,3,4, 2020.

[52] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2017.

[53] Gurucharan M K. Machine learning basics: Decision tree regression, 2020. `https://towardsdatascience.com/machine-learning-basics-decision-tree-regression-1d73ea003fda` [Accessed: 2024-04-18].

[54] Pooya Khorrami, Tom Le Paine, Kevin Brady, Charlie Dagli, and Thomas S. Huang. HOW DEEP NEURAL NETWORKS CAN IMPROVE EMOTION RECOGNITION ON VIDEO DATA. *arXiv*, 2017.

[55] Will Koehrsen. Introduction to bayesian linear regression. `https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7` [Accessed: 2024-04-18].

[56] Agata Kolakowska, Wioleta Szwoch, and Mariusz Szwoch. A Review of Emotion Recognition Methods Based on Data Acquired via Smartphone Sensors . *sensors*, pages 19–21, 2020.

[57] Dimitrios Kollias and Stefanos Zafeiriou. Affect analysis in-the-wild: Valence-arousal, expressions, action units and a unified framework, 2021.

[58] Shashidhar G. Koolagudi and K. Sreenivasa Rao. Emotion recognition from speech: a review. *Springer Link*, page 16, 2012.

[59] Jean Kossaifi, Georgios Tzimiropoulos, Sinisa Todorovic, and Maja Pantic. AFEW-VA database for valence and arousal estimation in-the-wild. *Elsevier*, 2017.

[60] Jean Kossaifi, Robert Walecki, Yannis Panagakis, Jie Shen, Maximilian Schmitt, and et al. SEWA DB: A Rich Database for Audio-Visual Emotion and Sentiment Research in the Wild. *arXiv*, 2019.

[61] Anders Krogh. What are artificial neural networks? *ResearchGate*, 2008.

[62] Sze Chit Leong, Yuk Ming Tang, Chung Hin Lai, and C.K.M. Lee. Facial expression and body gesture emotion recognition: A systematic review on the use of visual data in affective computing. *Computer Science Review*, 48:100545, May 2023.

[63] Liying Liu and Yain-Whar Si. 1D convolutional neural networks for chart pattern classification in financial time series. *Springer Link*, 2022.

[64] Foo Jia Ming, Shaik Shabana Anhum, Shayla Islam, and Kay Hooi Keoy. Facial emotion recognition system for mental stress detection among university students. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6, 2023.

[65] Gregoire Montavon, Genevieve B. Orr, and Klaus-Robert Mueller. *Neural Networks: Tricks of the Trade (Chapter 2)*. Springer, 2012.

[66] Eduardo Muñoz. Attention is all you need: Discovering the transformer paper.

[67] Andrew Ng. Kernel Methods. *Stanford Edu*, pages 1–11, 2019.

[68] Arnaud Nguembang Fadja, Evelina Lamma, and Fabrizio Riguzzi. Vision Inspection with Neural Networks. *ResearchGate*, 2018.

[69] Peter Norvig and Stuart J. Russell. *Artificial Intelligence: A Modern Approach*. Pearson, 2010.

[70] Deogratias Nurwaha. Comparison of kernel functions of support vector machines: A case study for the solar cell output power prediction. *Editorial Board Members of IJEAT*, page 3, 2020.

[71] Aaryan Ohekar. What is the difference between a decision tree classifier and a decision tree regressor?, 2023. `https://medium.com/@aaryanohekar277/what-is-the-difference-between-a-decision-tree-classifier-and-a-decision-tree-regressor-36641bd6559c` [Accessed: 2024-04-18].

[72] Art B. Owen. A robust hybrid of lasso and ridge regression. *Semantic Scholar*, pages 1–3, 2006.

[73] R Ghana Praveen, Wheidima Carneiro de Melo, Nasib Ullah, Haseeb Aslam, and Theo Denorme. A Joint Cross-Attention Model for Audio-Visual Fusion in Dimensional Emotion Recognition. *IEEE*, pages 1,2,8, 2022.

[74] Haonan Qui, Liang He, and Feng Wang. Dual Focus Attention Network For Video Emotion Recognition. *IEEE*, pages 1,6, 2020.

[75] Mickael Rouvier, Pierre-Michel Bousquet, and Jarod Duret. Study on the temporal pooling used in deep neural networks for speaker verification. *arXiv*, 2021.

[76] Mohd Saqib. Forecasting COVID-19 outbreak progression using hybrid polynomial-Bayesian ridge regression model. *Springer Link*, 2020.

[77] Iqbal H. Sarker. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *Springer Link*, 2021.

[78] Alakh Sethi. Support vector regression tutorial for machine learning, 2024. `https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/` [Accessed: 2024-04-18].

[79] Himani Sharma and Sunil Kumar. A Survey on Decision Tree Algorithms of Classification in Data Mining. *ResearchGate*, 2016.

[80] Alex Shenfield and Martin Howarth. A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults. *ResearchGate*, 2020.

[81] Hyunuk Shin, Bonhwa Lee, and Hanseok Ko. Noisy label facial expression recognition via face-specific label distribution learning. *Science Direct*, pages 1–2, 2024.

[82] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru, 2023.

[83] Rebecca C. Steorts. Tree based methods: Regression trees. `https://www2.stat.duke.edu/~rcs46/lectures_2017/08-trees/08-tree-regression.pdf` [Accessed: 2024-04-18].

[84] S. Suganya and E. Y. A. Charles. Speech Emotion Recognition Using Deep Learning on audio recordings. *IEEE*, pages 1,2, 2019.

[85] Tamanna Sultana, Jahan Meskat, Md.Kamal Uddin, Yoshinori Kobayashi, and Mahmudul Hasan. Multimodal Emotion Recognition through Deep Fusion of Audio-Visual Data. *IEEE*, pages 1,5, 2023.

[86] Licai Sun, Bin Liu, Taom Jianhua, and Zheng Lian. Multimodal Cross- and Self-Attention Network for Speech Emotion Recognition. *IEEE*, pages 1,4, 2021.

[87] Abhishek V Tatachar. Comparative Assessment of Regression Models Based On Model Evaluation Metrics. *IRJET*, 2021.

[88] udit. Activation functions in deep learning: Understanding the role of activation functions in neural networks. `https://itsudit.medium.com/activation-functions-in-deep-learning-understanding-the-role-of-activation-functions-in-neural-423694f7f54e`[Accessed: 2024-04-18].

[89] Avadhut Varvatkar. Ridge regression. `https://www.kaggle.com/code/avadhutvarvatkar/ridge-regression` [Accessed: 2024-04-18].

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[91] Viswa. Support vector regression: Unleashing the power of non-linear predictive modeling, 2023. `https://medium.com/@vk.viswa/support-vector-regression-unleashing-the-powerof-non-linear-predictive-modeling-d4495836884`[Accessed: 2024-04-18].

[92] Lipo Wang and V. Kecman. *Support Vector Machines: Theory and Applications. Chapter: Support Vector Machines – An Introduction.* Springer Link, 2005.

[93] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer, 2004.

[94] Wessel N. van Wieringen. Lecture notes on ridge regression. *arXiv*, 2023.

[95] Drew Wilimitis. The kernel trick in support vector classification, 2018. `https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f` [Accessed: 2024-04-18].

[96] Jinchang Xu, Yuan Dong, Lilei Ma, and Hongliang Bai. Video-based Emotion Recognition using Aggregated Features and Spatio-temporal Information. *IEEE*, pages 1,2,4, 2018.

[97] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Dao, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Springer Open*, 2018.

[98] Muhamad Yani and S Si. M.T. Budhi Irawan. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. *ResearchGate*, 2019.

[99] Xue Ying. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 2019.

[100] Liang-Chih Yu, Lung-Hao Lee, Shuai Hao, Jin Wang, Yunchao He, Jun Hu, K. Robert Lai, and Xuejie Zhang. Building chinese affective resources in valence-arousal dimensions. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2016.

[101] Siyuan Zheng, JUn Du, Hengshun Zhou, Xue Bai, and Shipeng Li. Speech Emotion Recognition Based on Acoustic Segment Model. *IEEE*, pages 1,4, 2021.