OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Data Mining at Scale: Algorithms and Systems
ACIT4530

Project report

EEG emotion classification – LSTM and GRU models

Authors: Alexandros Messaritakis C(S326156), Lars Storholt (s354518), Kristian Jørgensen (s344189) **Date: 24.05.2024**

Abstract

This project aims to compare how Long short-term Memory (LSTM) and Gated Recurrent Network (GRU) models detect emotions in Electroencephalogram (EEG) signals to ascertain if comparative accuracy of LSTM can be achieved with the computationally cheaper model GRU. Both models performed well and as expected the LSTM model performed better than the GRU model. In comparing the two models, it is clear that the LSTM model offers superior accuracy, making it the preferred choice for applications requiring precise emotion classification from EEG signals. However, the GRU model's performance is noteworthy, especially considering its reduced computational requirements. This makes the GRU model a practical alternative in scenarios where computational resources are limited. In this project the GRU model has proved to be slightly inferior to the LSTM model in terms of accuracy as shown in Table 1, however the Gru model performed well enough to be a real alternative to LSTM.

1. Introduction

The brain is a complex and powerful organ that can be likened to a computer. It has trough many science fiction novels and stories come a notion that many of the universe secrets are locked away inside and that by understanding the brain can unlock its potential. It is in this spirit that we hope to make a dent in the unlocking of the mind’s secrets. Furthermore, understanding the mind better can aid people who are suffering either from a mental disease, disorder, and perhaps many of the common ails that is experienced by the general populous.

Electroencephalogram (EEG) detects brainwaves and record them. To the untrained eye the readouts look like frequential noise, but there is information stored in the data readouts. This project aims to train a neural network (NN) towards discerning emotions based on the EEG readouts. These readouts are taken of people watching clips from: “Marley and me”, “Up”, “My girl”, “La La Land”, “Slow life”, and “Funny dogs” which consists of various genres ranging from comedy, horror, and funny videos of the internet. Ideally the NN would be able to detect the exact emotion felt by the participant, however in this project it will be sufficient to determine whether the feeling is positive, neutral, or negative. The positive and negative emotions are responses to clips whereas the neutral emotions are from time with no video-clips. There are two batches of three minutes one for positive and one for negative. There was also a neutral section without any stimuli lasting a total of six minutes The test participants were one male and one female

The main method of discerning emotions in this project is Long short-term memory (LSTM), a variant of Recurrent Neural Network (RNN). RNNs are good at processing shorter sequences of data and does this very well. However, RNNs struggle with longer sequences of data due to a vanishing gradient problem. LSTMs are able to counter this problem during training allowing it to handle longer sequences of data.

Compare GRU with LSTM and see if the same result can be achieved cheaper computationally.

2. Background

2.1 Introduction to the topic

EEG signals are used to discern the state of the human mind by taking factors such as valence, dominance and arousal. Anticipating the preferences of users on their emotional state at the time they engage with specific applications is crucial. This foresight allows for the presentation of content that is not only relevant but also unique and satisfying. As technology progresses and innovative devices capable of capturing emotions—like EEG machines – become available, individuals will have the opportunity to consistently receive tailored content, enhancing their experience with applications centred around content consumption.

2.2 EEG: Deciphering the Brain's Electrical Narratives

According to the Mayo Clinic (n.d.) an electroencephalogram (EEG) is a technique for tracking and recording brain wave patterns. It is fundamentally a collection of the brain's spontaneous electrical activities, captured by electrodes placed on the scalp. These activities primarily reflect synchronous activity of neuronal ensembles, especially in the brain's surface layers. EEG is known for its non-invasive nature, meaning that there is no need for surgery but instead employing electrodes that rest on the scalp that are used to pick up electrical signals. These signals are then amplified and recorded, proving valuable insights into brain function and health.

EEG's utility covers several areas within neurology and psychiatry, including the diagnosis and management of epilepsy, where it is instrumental in detecting abnormal electrical discharges indicative of seizures. In addition, it plays a role in assessing conditions like sleep disorder, evaluating coma depths, and monitoring cerebral states following significant events like cardiac arrest. Despite the advent of advanced imaging modalities like MRI and CT scans, EEG holds its ground due to its unique capability to offer real-time insights into brain activity with high temporal resolution.

Moreover, EEG technology has expanded to include ambulatory systems, allowing for extended monitoring in non-clinical settings, thereby expanding the possibilities for capturing and analysing brain activity across different states and conditions.

2.3 DEAP Database: Bridging Emotions with Technology

The DEAP database is described as per the paper from Sander Koelstra et.al (2012) as a multimodal dataset for the analysis of human affective states, particularly focusing on the emotional elicited by music videos. The dataset records the electroencephalogram (EEG) and peripheral physiological signals from 32 participants as they watched 40 one-minute excerpts of various music videos. Participants were asked to rate each video on several dimensions, including arousal, valence, like/dislike, dominance and familiarity. In some cases, frontal face video recordings were also made for a subset of participants. The DEAP dataset utilized a unique method for selecting stimuli, employing a combination of affective tags from the Last.fm website, video highlight detection and an online assessment tool to choose music video that would evoke a broad spectrum of emotional responses. This dataset is made publicly available to encourage further research in affective computing, particularly in the development and testing of new methods for emotion analysis.

2.4 The Emotional Spectrum Through EEG: Valence, Arousal, Dominance

In the context of the DEAP paper, the state of the human mind, particularly in relation to emotions, is conceptualized along the dimensions of valence, arousal, and dominance. These dimensions are part of Russell's model of affect, which is widely used in emotion research. This model proves that every emotional state can be positioned on a two-dimensional plane, with arousal representing the level of alertness or excitement, ranging from low (e.g., calm) to high (e.g., excited), and valance indicating the positivity or negativity of the emotional state, ranging from negative (e.g., unhappy) to positive (e.g., happy). The addition of dominance introduces a spectrum from feeling controlled or submissive to feeling in control or dominant. The paper written by Sander Koelstra et.al (2012) employs this model to quantify and assess emotional responses to stimuli, in this case, music videos, by having participants rate their reactions in terms of these three dimensions using self-assessment manikins (SAM).

2.5 Activation function

There are many activation functions, each tailored to certain problems and models. The activation functions work as gate keepers deciding what value to let through to the next step and what to not pass. This is achieved through a threshold which the value must clear, thus activating the function.

Sigmoid

Any function forming an S shape can be called a sigmoid function, in regard to neural networks however, the sigmoid activation function or just Sigmoid function is a mathematical logistical function defined as:

$$g(z) = \frac{1}{1+e^{-z}}$$

The sigmoid function is good for classification of data due to the function returning a value between 0 and 1.

Tanh

The tanh activation function is commonly used in the hidden layers of a neural network. Hidden layers are referring to the layers of nodes between the input (first) layer and output (last) layer. It operates between –1 and 1 giving the average of 0. This helps to prevent the values becoming very large or very small causing what is known as vanishing gradient which is a particular problem for RNNs of which LSTMs are a variant. The tanh activation function is defined as:

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

SoftMax

SoftMax activation function takes input values and turn them into values between 0 and 1, making it similar to the sigmoid function. It does however have slightly different applications and have some conditions in order to function as a classifier. The general function is given as:

$$g\left(\vec{z}\right)i = \frac{e^{zi}}{\sum_{j=1}^{K} e^{zj}}$$

2.6 Recurrent Neural Network (RNN)

RNNs are generally good for sequential data, meaning data that is dependent on the data that came before in the sequence. The way it achieves this is by injecting the output of the previous cell into the current cell. This means that the RNN retains information from previous outputs through the current. Providing a good ability for language and EEG, however the downside is that it expends memory quickly causing it to struggle if the sequence becomes too long.

2.7 Long Short-Term Memory (LSTM)

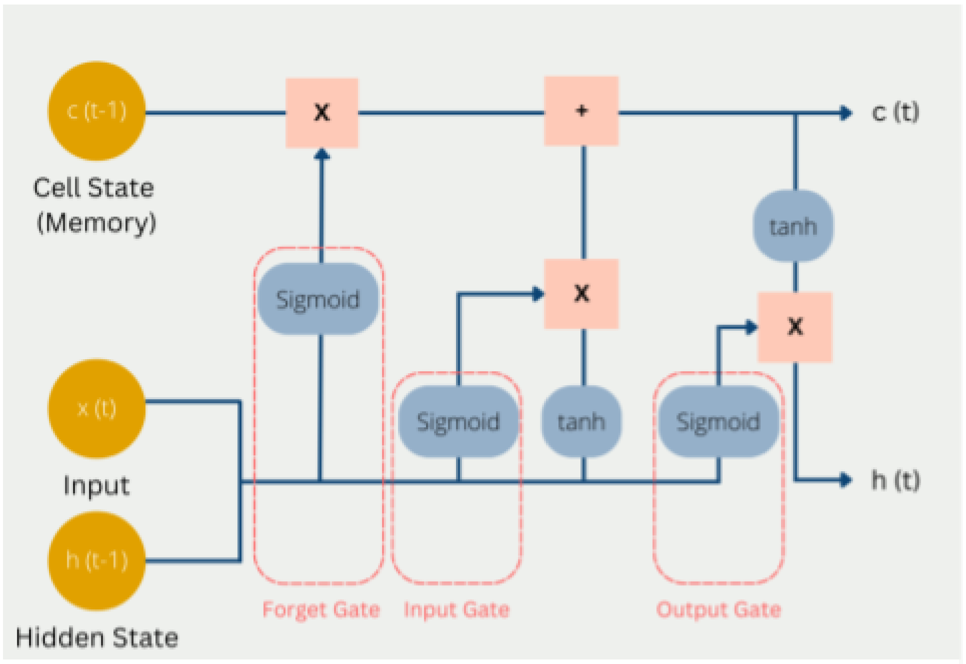


Figure 1: LSTM Flow diagram. Ref: https://databasecamp.de/en/ml/lstms

LSTM networks feature a more complex architecture compared to traditional RNNs, with the use of gated mechanism. These gates serve to discern and prioritize elements within the input data, by dynamically modifying the content stored in the memory cell. The input gate ranks the importance of each input element, the forget gate decides which information in the memory cell is retained or forgotten, and the output gate controls what of current memory state is to be output for the subsequent step. Enhanced by the sigmoid and tanh activation functions, manage to filter out important and non-important information from the data. This capability makes LSTMs capable of managing longer sequences of data and handling long-term dependencies.

2.8 Gated Recurrent Unit (GRU)

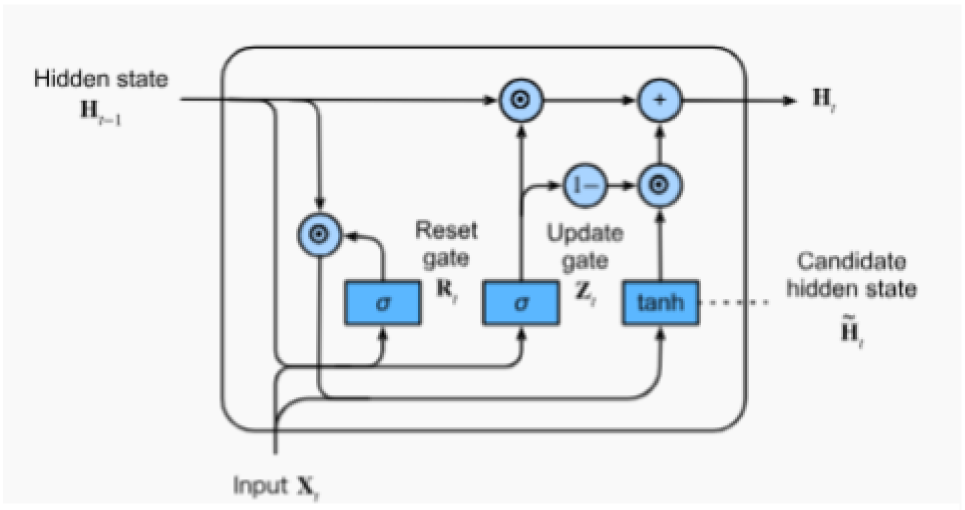


Figure 2: GRU flow diagram (Jameson, 2021)

A GRU neural network, like the LSTM, is a type of recurrent neural network, which uses gated mechanism and memory cells for better understand long sequence data. In comparison with the LSTM, which utilizes three gates, the GRU architecture is less complex, featuring only two gates: The reset gate determines the proportion of information from the previous states that are to be forgotten. Conversely, the update gate controls the amount of the previous steps information to be sent to the future hidden layers. The simplicity of the GRU-architecture facilitates faster training. This can also lower GRUs precision on longer sequential data compared to LSTMs. Despite that, for medium- to short-length data, GRU has demonstrated superior performances in some cases. (Harsha, 2023)

2.9 Convolutional Neural Network (CNN)

An alternative model to the RNNs, which have shown effectiveness with EEG data, is a Convolutional Neural Network (CNN). Unlike the RNNs, CNNs perceive the EEG data more like a 2D image rather than a sequence of data. In this approach, the time dimension represents the width of the image, while the EEG signals represent the height. Previous research has demonstrated great classification results using this approach. For instance, the study titled “A Simplified CNN Classification Method for MI-EEG via the Electrode Pairs Signals”, achieved an accuracy of 97.28% in a four-class classification problem using CNN.

2.10 Transformer Model

The transform model, gaining a lot of traction these days, with high preforming models like Chat GPT. It has achieved impressive performance with lengthy data sequences, through self-attention mechanism. Here, the model determines the importance or relevance by comparing the input elements with every other element.

3. Methodology

The code used in this project is stored on github, and is available through the link in the references section.

3.1 Dataset

The dataset “Emotions.csv” originates from a project conducted at Aston University, Birmingham. It comprises EEG data recorded from a male and a female participant while they watched movie clips designed to evoke distinct responses.

The total duration of the video was 36 minutes, segmented into eight smaller scenes from different movies. The purpose of the clips was to invoke either positive, neutral, or negative emotions states in the participants.

EEG sensors were positioned at TP9, AF7, AF7, and TP10 locations, within the international EEG standard placement.

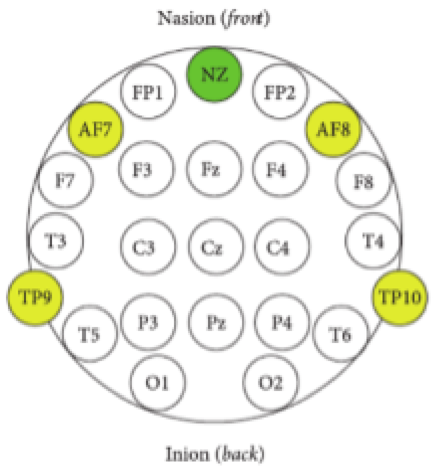


Figure 3: EEG sensor placement (J. Bird et al., 2019)

The original dataset consisted of 324,000 datapoints, representing each EEG sensor measure at a specific timestamp. To enhance its suitability for machine learning training, the data underwent preprocessing using various statistical methods such as covariance calculation, min/max normalization, Fast Fourier Transformation, and mean calculation. This preprocessing was applied to EEG samples within a 1-second window. As a result, the dataset was expanded to 2549 features and subsequently reduced to 2132 rows. (J. Bird et al., 2019) The dataset is balanced, with an equal duration of movie clips for each category, resulting in 710 datapoints for each emotion state.

4. Implementation

4.1 Overview

This script uses a dataset of EEG recordings to train a Long Short-Term Memory (LSTM) - and a Gated Recurrent Unit (GRU) network, aimed at classifying emotions into three categories: 'Negative', 'Neutral', and 'Positive'. The process involves loading data, preprocessing it, splitting it into training and testing sets, building an LSTM- and the GRU models, and finally, evaluating the models' performances using a confusion matrix.

4.3 Data Preprocessing

```
features = [col for col in data.columns if 'fft' in col]
X = data[features].values
```

FFT, or Fast Fourier Transform, is a method to compute the Discrete Fourier Transform (DFT) and its inverse. FFT is widely used in signal processing to analyse the frequencies contained within a signal, and it's particularly useful in EEG data analysis because different emotional or cognitive states can be associated with different frequency bands (such as alpha, beta, gamma). By focusing on 'fft' features, the model is likely being directed to learn from the most significant aspects of the EEG data related to frequency analysis. These features might represent power spectra, which are indicative of how power (variance) in the data distributed across different frequency components.

EEG signals are typically analysed either in the time domain or frequency domain. The frequency domain provides insights into the oscillatory activity of the brain, which is crucial for studying brain states related to different emotions. By transforming EEG signals into the frequency domain using FFT, the data becomes more manageable and often more indicative of the physiological and psychological states being studied. For instance, specific frequency bands are known to be associated with different types of brain activity and states.

The features chosen for the model input are specifically those containing the substring 'fft' in their column names. This suggests a focus on the frequency domain features of the EEG data, which are extracted using Fast Fourier Transform (FFT). FFT is a mathematical technique that transforms time-domain data into frequency-domain data, revealing the periodic structures and spectral components of the EEG signals. These features are important for analysing EEG because different emotional states may be characterized by distinct frequency patterns.

Next, we move to normalizing the data with Standard Scaler. This step involves scaling the features to have a zero mean and a unit variance. In practice, this involves subtracting the mean and dividing by the standard deviation for each feature. Normalizing the data is vital because LSTM units include sigmoid and tanh activation functions that can be sensitive to the magnitude of inputs. Scaled data help in speeding up the convergence during training and prevent the model from being biased toward features with inherently large magnitudes or variances.

```
 scaler = StandardScaler()
 X_scaled = scaler.fit_transform(X)
```

Here, `fit_transform()` computes the mean and standard deviation that are required for scaling the data and then automatically applies this transformation. The scaler object holds the mean and standard deviation for each feature to allow the same transformation to be applied to new data during model testing.

4.4 Reshaping Data for LSTM & GRU Input

The models expect input in the form of 3D arrays, where the dimensions represent:

Samples: The total number of data points or observations in the dataset.
Time steps and features: The number of time intervals in each sample. In this script, each sample is treated as having a single timestep.

Typically, LSTM models are used to leverage temporal dynamics in data, meaning they perform best when each sample consists of sequences captured over multiple time steps. In this case, treating each FFT feature set as a single timestep suggests that either the temporal aspect is encapsulated within each feature, or this model architecture is an initial baseline to test LSTM applicability.

X_reshaped = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

This line of code reshapes the normalized feature matrix `X_scaled` into the required 3D structure by specifying that each sample has one timestep and retains the original number of features per timestep. This reshape is critical for matching the input expectations of the LSTM layer in the network. The preprocessing steps transform raw EEG data into a format that is suitable for training an LSTM network, focusing on ensuring that the input features are normalized and appropriately structured to highlight potential patterns associated with different emotional states as captured through EEG signals. This methodical preprocessing is designed to optimize the neural network's learning process and performance.

4.5 Encoding Labels and Data Splitting

The journey from raw data to a trainable format begins with the encoding of labels. In our model's setup, we encounter a scenario common in many machine learning tasks: the need to convert categorical labels into a machine-readable format. This is achieved through **one-hot encoding**

y_encoded = pd.get_dummies(y).values

One-hot encoding is pivotal because it translates the categorical labels—'Negative', 'Neutral', and 'Positive'—into a binary matrix. This matrix format is not just a requirement but a boon for neural networks, which thrive on numerical input. Each label is represented by an array where one element is set to '1' (indicating the presence of the label) and all others to '0'. This encoding scheme aligns perfectly with the softmax activation function used later in the network, preparing the stage for efficient computation of probabilities across the multiple classes during training. Moreover, it simplifies the application of the categorical cross entropy loss function, which quantitatively measures the difference between the predicted probabilities and the actual label distribution.

Following the encoding, the next strategic step is the splitting of the dataset, which forms the basis for a robust evaluation of the model's performance:

X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y_encoded, test_size=0.2, random_state=42)

The train-test split serves a dual purpose. Primarily, it ensures that the model is trained on a diverse subset of data (80% in this case), providing it ample examples from which to learn. The remaining 20%, the test set, is crucial as it acts as new, unseen data for evaluating how well the model generalizes beyond its training data. The choice of an 80/20 split is a balanced approach, allowing enough data for learning while reserving a substantial subset for unbiased performance evaluation. Setting a random_state ensures that this split is reproducible, making the model's performance consistent across different runs and thus reliable for further analysis and tuning. Together, the encoding of labels and the splitting of data encapsulate a fundamental workflow in preparing data for neural networks. They not only facilitate the operational requirements of neural network architectures but also set the stage for training a model that can be rigorously tested and evaluated for its predictive prowess. This meticulous preparation ensures that the transition from data preparation to model training is seamless, paving the way for building a model that is as effective in theory as it is in practice.

4.6 Model Building

The construction of the LSTM and GRU models is a critical step that involves defining the layers, neurons, and activation functions that will operate on the input data. The models' architecture is tailored to process sequential data, making it ideally suited for EEG signals which are inherently time-series data.

4.6.1 Building the LSTM model

The model begins with the instantiation of a Sequential object from Keras.

model = Sequential()

This Sequential model is aptly named as it allows the straightforward creation of a linear stack of layers, where each layer has exactly one input tensor and one output tensor. It's an intuitive way to build models layer by layer. The first layer in the model is an LSTM layer.

LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2]))

Long Short-Term Memory units are a type of recurrent neural network (RNN) suitable for sequence prediction problems. This layer is designed to remember information for long periods, which is key in handling the dependencies seen in time-series data like EEG signals. The layer comprises 50 LSTM cells. This number is a balance between model complexity and computational efficiency. Each unit can capture a different aspect of the data, contributing to a more robust model. The input shape specified as (X_train.shape[1], X_train.shape[2]) corresponds to the structure of each input sequence. This parameter is crucial as it informs the LSTM layer about the dimensionality of each timestep and the number of features to expect.

To enhance the model's ability to generalize and prevent overfitting, a Dropout layer follows the LSTM.

Dropout(0.2)

This layer randomly sets input units to 0 with a frequency of 20% at each step during training time, which helps to prevent overfitting. Dropout is an effective and simple regularization technique that improves the robustness of neural networks, particularly in complex models that have a high risk of overfitting. The final layer in the model is a Dense layer, which is a fully connected neural network layer

4.6.2 Building the GRU model

GRU(256, return_sequences=True)(input_gru)

This layer consists of 256 GRU cells. Compared to the 50 LSTM cells, the higher number of cells has the potential to make the model learn more complex patterns. The simpler architecture compared to the LSTM contributes to its higher computational efficiency. Therefore, increasing the number of cells in the GRU model does not lead to slower training time.

Flatten()(x_gru)

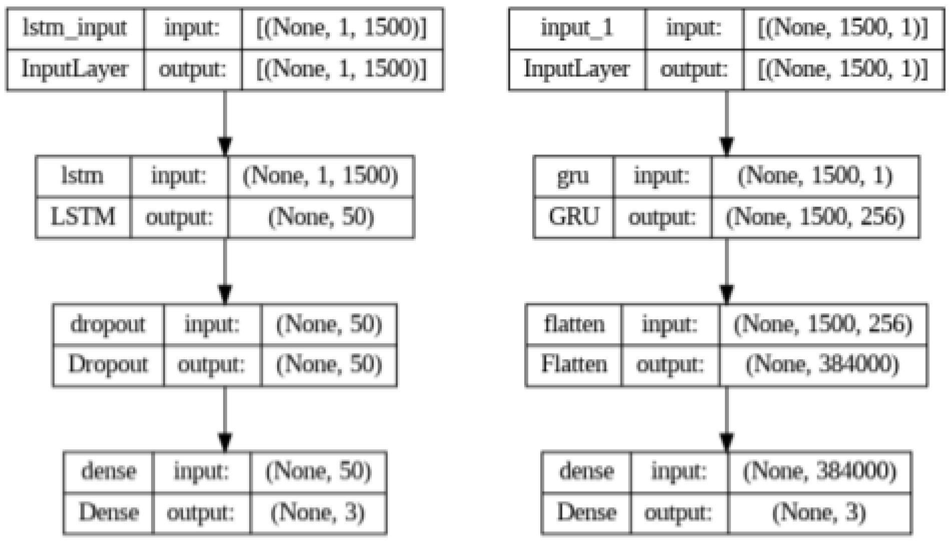
The flatten layer transforms the multidimensional output from the GRU layer into a one-dimensional tensor. The step is necessary for connecting the GRU layer to the following Dense layer.

Both the models conclude with is a dense layer.

Dense(y_encoded.shape[1], activation='softmax')

This layer outputs a vector where each entry corresponds to the probability of a class. The number of neurons in this layer equals the number of classes (y_encoded.shape[1]), which are 'Negative', 'Neutral', and 'Positive'. The SoftMax function is used because it is ideal for multi-class classification tasks. It transforms the outputs into probability scores that sum to one, which directly corresponds to the likelihood of each class given the input features.

By constructing the models with these layers, we ensure a smooth flow from data input through the LSTM's and GRU's complex transformations to output predictions. The models' architecture is thus not just a collection of layers, but a carefully thought-out system designed to handle the intricacies of EEG data, transforming raw inputs into actionable insights in the form of emotion classifications. This setup directly feeds into the subsequent training process, where the model learns from the training data by adjusting its weights to minimize the error in its predictions, evaluated against the actual labels during the training epochs.



lstm_input | input: | [(None, 1, 1500)]
InputLayer | output: | [(None, 1, 1500)]
lstm | input: | (None, 1, 1500)
LSTM | output: | (None, 50)
dropout | input: | (None, 50)
Dropout | output: | (None, 50)
dense | input: | (None, 50)
Dense | output: | (None, 3)
input_1 | input: | [(None, 1500, 1)]
InputLayer | output: | [(None, 1500, 1)]
gru | input: | (None, 1500, 1)
GRU | output: | (None, 1500, 256)
flatten | input: | (None, 1500, 256)
Flatten | output: | (None, 384000)
dense | input: | (None, 384000)
Dense | output: | (None, 3)

Figure 4: Architecture of GRU- and LSTM – model

The images show the GRU and LSTM architectures. The LSTM design comprises an input layer that reduces 1500 input features to 50 output features, a dropout layer, and a dense layer that outputs the three features. The appropriate design begins with a GRU layer that converts 1500 input features into 256 output features, which then travel through a flatten layer to generate 384000 features, and concludes with a dense layer.

4.7 Model Compilation and Training

Compiling the models configures it for training by specifying the loss function, the optimizer, and any metrics to be monitored during the training and evaluation processes.

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

The Adam optimizer is chosen for its efficiency in handling large datasets and high-dimensional spaces. Adam stands for Adaptive Moment Estimation; it combines the advantages of two other extensions of stochastic gradient descent. Specifically, it computes adaptive learning rates for each parameter from estimates of first and second moments of the gradients. This makes it particularly effective for problems with noisy or sparse gradients, a common scenario in neural network training, including LSTM and GRU networks.

This loss function is appropriate for multi-class classification problems like the one being tackled here. Categorical cross entropy measures the disparity between the predicted probability distribution (output of the softmax function in the last layer) and the true distribution, where the true distribution is the one-hot encoded labels of the training data. The goal during training is to minimize this loss, which in turn maximizes the accuracy of the predictions against the actual labels.

This metric is used to monitor the training and testing steps. It gives a straightforward indication of the model's performance, calculated as the ratio of the number of correct predictions to the total number of input samples. It's particularly useful for classification problems to provide insight into the percentage of correctly classified instances.

```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

This method adjusts the weights of the network to minimize the loss function over a specified number of iterations through the dataset, referred to as epochs. Epochs are the number of times the entire dataset is passed forward and backward through the neural network. More epochs can lead to better learning, but at the risk of overfitting if not monitored carefully. Next, we have the batch size which defines the number of samples that will be propagated through the network in one forward/backward pass. A smaller batch size provides a more fine-grained update to the weights, while a larger batch size offers computational advantages in terms of speed. Finally, we have the validation data which guides the training direction if adjustments need to be made and ensures that the model's performance improves not just on the training data but generalizes well to new data. This is useful if we want to monitor the model's performance on unseen data.

4.8 Model Evaluation and Visualization

The evaluation of the models is done using the test dataset, which was set aside during the data splitting stage. This step is crucial for assessing the models' generalization capabilities.

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

This function calculates the loss and accuracy of the model using the test data, providing a quantitative measure of its performance. The loss indicates how well or poorly a specific model behaves after each optimization iteration over the test data, whereas accuracy gives a more direct understanding of its effectiveness, specifically:
The loss variable reflects the model's errors during prediction, with lower values indicating better performance while the accuracy variable represents the percentage of correct predictions out of all predictions made, a key indicator of success, especially in classification tasks. This quantitative evaluation helps validate the model's learning and predictive accuracy but does not provide detailed insights into the nature of any misclassifications or the model's behaviour across different classes. To further understand the model's performance, particularly how well it predicts each class and where it makes errors, a confusion matrix is generated and visualized.

```
predicted_probabilities = model.predict(X_test)
predicted_labels = np.argmax(predicted_probabilities, axis=1)
true_labels = np.argmax(y_test, axis=1)
cm = confusion_matrix(true_labels, predicted_labels)
```

The model's predict method is used to get the predicted probabilities for each class. The np.argmax function then converts these probabilities into actual class predictions by selecting the class with the highest probability. This matrix is a powerful tool for categorical analysis, showing the correctly and incorrectly predicted instances in a format that's easy to understand. Each row of the matrix represents the instances in an actual class, while each column represents the instances in a predicted class. The diagonal cells represent the number of correct predictions, which ideally should be high, and the off-diagonal cells represent misclassified instances.

```
plt.figure(figsize=(10, 7))
ax = sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
yticklabels=labels, vmin=0)


for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j+0.5, i+0.5, cm[i, j],
                ha="center", va="center",

                color=get_annotation_color(cm[i, j]))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

The heatmap from Seaborn is used to display the confusion matrix, providing a visual representation of the model's performance across different classes. The colour intensity in the heatmap reflects the number of instances, making it easier to identify where the model performs well and where it does not. Text annotations added to the heatmap improve its readability, ensuring that each cell's count is clearly visible regardless of its colour.

5. Results

5.1 Training results

The LSTM model trained to classify emotions from EEG data, showed good performance from the start. Initially, it achieved a training accuracy of about 93.06% and a validation accuracy of 92.27%. This early success indicates that the model quickly learned from the training data.

As the training progressed, both the training and validation accuracies steadily improved. By the tenth training epoch, the model reached a training accuracy of 96.77% and a validation accuracy of 96.49%. This suggests that the model was not only learning effectively but was also able to apply what it learned to new, unseen data.

In the later stages of training, the accuracy continued to increase slowly, showing that the model was optimizing its performance. By the end of training at epoch 20, the training accuracy was about 97.42% with the validation accuracy around 96.25%. Along with the increase in accuracy, the loss — which measures errors — decreased significantly from the start of training to the end. This reduction in loss confirmed that the model's predictions were becoming more accurate over time.

The consistent high validation accuracy and low validation loss across all epochs showed that the model was learning to generalize, meaning it could perform well not just on the data it was trained on but also on new data it hadn't seen before. This ability led to suggesting that the model is reliable and can be used effectively in real-world settings, in our case predicting the user's emotions.

The consistent performance on the validation set reassures us that the model is robust and can handle new data effectively. This makes it a useful tool for further research and applications in detecting emotions from EEG signals. Such performance also reflects well on the design of the model and the methods used to train it, suggesting that the choices made in setting up the model were appropriate for the task.

5.2 Testing results

The testing results from the LSTM model show a high level of accuracy in identifying the correct emotions from EEG data. Specifically, the model achieved a test accuracy of 96.2%, which is a strong indication of its effectiveness. This performance metric reflects how well the model can apply what it has learned during training to a new set of data, which in this case involves correctly classifying emotions as negative, neutral, or positive.

A test accuracy of over 96% is particularly impressive and shows that the model is well-tuned and robust. It implies that the features extracted from the EEG data, the architecture of the LSTM model, and the training process—including the handling of data normalization and the configuration of layers—were all well-designed for the emotion classification.

This high level of accuracy on test data is important as it demonstrates the model's ability to generalize beyond the examples it was trained on. This is important in practical applications where the model will encounter data variations that were not present in the training set. The LSTM model's test results are a strong validation of its design and training approach, showing that it can accurately decode the emotional states from EEG signals, making it a valuable tool for studies and applications that require emotion recognition from neurological data.

Table 1: Score of models' performance

Model	Accuracy	Precision	Recall	F1-score
LSTM	96,2 %	96,3 %	96,2 %	96,2 %
GRU	93,2 %	93,3 %	93,3 %	93,3 %

5.3 Confusion Matrix Graph

The confusion matrix for the models provides a comprehensive look at its performance in classifying emotions from EEG data into three categories: Negative, Neutral, and Positive. By examining each cell in the matrices, we gain insights into the accuracy of the model's predictions and the types of errors it tends to make.

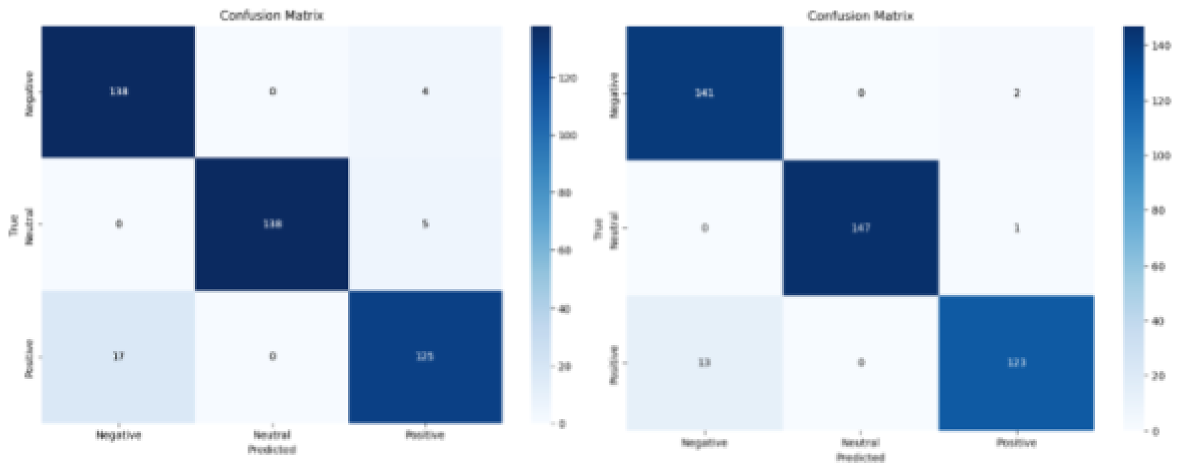


Figure 5: Confusion matrices of both the LSTM- and GRU model. (Left: GRU, Right: LSTM)

Starting with Negative emotions, the models show a strong capability, correctly identifying 141 out of 143 instances for the LSTM model and 138 out of 142 of the GRU model. This high accuracy indicates the models' robustness in recognizing negative emotional states from EEG signals. However, it's not without its shortcomings, as evidenced by the two and four instances where Negative emotions were mistakenly classified as Positive, suggesting a slight confusion between these two emotional states.

Moving to Neutral emotions, the LSTM model's performance is nearly flawless, with 147 out of 148 instances accurately classified. Only a single Neutral emotion was misclassified as Positive, underscoring a rare but present challenge in distinguishing between Neutral and Positive emotions. This minor error rate points to the model's effectiveness but also hints at potential areas for fine-tuning.

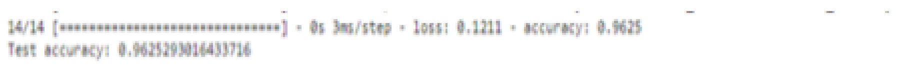
The GRU model accurately predicted the neutral class in 138 out of 143 instances. In all cases of incorrect predictions, it misclassified them as positive, performing slightly worse than the LSTM model

The classification of Positive emotions, however, revealed more significant challenges. The models incorrectly classified 13 (LSTM) and 17 (GRU) instances of Positive emotions as

Negative, which is the most notable error observed in the matrix. This suggests that the models might struggle with overlapping characteristics in EEG signals that represent Positive and Negative emotions or that additional features are needed to enhance discrimination between these two states.

Despite these errors, the models correctly identified 123 and 125 instances of Positive emotions, demonstrating a good level of accuracy for this category. The predominance of correct classifications across all three emotional states, as seen in the significantly higher diagonal values compared to the off-diagonal ones in the matrix, confirms the model's overall effectiveness.

This analysis not only highlights both the models' strengths in accurately classifying emotions but also illuminates specific areas where improvements could be made, particularly in differentiating Positive from Negative emotions. The LSTM model outperforms the GRU model slightly, achieving an accuracy of 96.3% compared to the GRU's 93.2%. Enhancing the models might involve adjusting parameters, incorporating more training data, or exploring new features that capture the nuances of emotional states more distinctly.



```
14/14 [==============================] - 0s 3ms/step - loss: 0.1211 - accuracy: 0.9625
Test accuracy: 0.9625293016433716
```

Figure 6: Test results of LSTM model show 96.2% accuracy of identifying the right emotions.

6. Discussion

6.1 LSTM/GRU vs. RNN vs. CNN for EEG Data Analysis

When comparing LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), RNN (Recurrent Neural Network), and CNN (Convolutional Neural Network) for EEG (Electroencephalography) data analysis, it's essential to understand the distinct capabilities and ideal applications of each model due to the unique nature of EEG signals.

LSTMs and GRUs are specifically engineered to address the long-term dependency challenge, allowing them to store information for extended periods without succumbing to the vanishing gradient problem often seen in traditional RNNs. This attribute is particularly valuable in EEG data analysis, where capturing long-duration temporal patterns is crucial for identifying phenomena such as seizures or sleep stages. LSTMs/GRUs manage to remember significant data points over time, making them excellent for tasks that not only require recognizing patterns but also recalling them much later, as demonstrated by Xu et al. (2020) in their study using a CNN-LSTM model for epileptic seizure recognition.

On the other hand, while RNNs share some similarities with LSTMs in processing sequences through their internal memory, they falter in learning long-range dependencies effectively. This limitation stems from the inherent problem of vanishing gradients during training, leading to diminished performance when dealing with lengthy data sequences. This makes RNNs less ideal for continuous EEG monitoring, where remembering extensive past information is crucial. However, their simplicity makes them suitable for shorter sequences or less complex EEG analysis tasks.

CNNs offer a different set of advantages. Known for their prowess in extracting patterns from spatial data, CNNs can identify spatial features across multichannel EEG data, pinpointing areas of the brain that show simultaneous activity. This ability is crucial for tasks like raw EEG signal processing, where the robustness against noise and variance in data is beneficial. The integration of CNNs with LSTMs, as seen in the work by Garcia-Moreno et al. (2020), leverages the spatial pattern recognition capability of CNNs alongside the sequence modelling strength of LSTMs. This combination is particularly effective for comprehensive EEG analysis that requires handling both spatial and temporal dimensions.

Choosing between these models typically depends on the specific EEG analysis requirements. LSTMs are preferred for complex temporal processing, CNNs are ideal for spatial feature extraction, and RNNs are applicable for simpler, shorter sequence tasks. Often, hybrid models, such as CNN-LSTM, provide the most robust solution by merging the strengths of CNNs and LSTMs to effectively manage the spatial and temporal challenges present in EEG data analysis. This approach has proven potent in several studies, underscoring the synergy between these architectures to enhance both the depth and accuracy of EEG signal interpretation.

6.2 Factors Influencing Model Choice

Models like LSTMs are intricate and require significant computational power due to their sophisticated feedback mechanisms designed to capture long-term dependencies. This makes them computationally expensive, especially when handling large datasets which are common in EEG analysis. However, simpler RNNs, while less demanding in terms of computational resources, may not perform as well on tasks that require learning from long data sequences. By combining simpler model architectures like GRU with LSTM, we can achieve a balance between computational economy and model performance.

The simplicity of GRU models compared to LSTMs, enables us to deploy models with more cells within the same computing budget, maximizing the model's ability to capture complicated patterns in EEG data. Despite their simplicity, GRUs are specifically designed to overcome the difficulty of long-term dependencies in sequential data, giving them a viable alternative to LSTMs for jobs requiring learning from lengthy sequences.

Accuracy is critical, particularly when the models are used in contexts like medical diagnosis from EEG data. LSTMs are often preferred for their superior ability to capture long-duration temporal patterns critical in EEG applications, such as monitoring epileptic seizures or analyzing sleep stages.

Meanwhile, CNNs are recognized for their capability to extract spatial features efficiently, making them invaluable for identifying spatial patterns across EEG channels. This is particularly useful in tasks that require recognizing areas of brain activity. Studies like those by Jiang et al. (2021) and Rim et al. (2020) illustrate how LSTM models surpass traditional RNNs in stability and accuracy, especially in handling EEG signals induced by specific stimuli, highlighting their suitability over simpler RNNs for complex EEG data analysis tasks.

The specific nature of the EEG analysis task heavily influences model selection. For instance, tasks that require detailed recognition of spatial patterns may benefit more from CNNs due to their strong feature extraction capabilities. In contrast, applications that rely heavily on understanding temporal dynamics are better served by LSTMs or hybrid models that combine the strengths of CNNs and LSTMs.

The type of EEG data can also dictate the choice of the model. LSTMs/GRUs are particularly effective in dealing with noisy, long-duration time-series data, thanks to their ability to manage variable sequence lengths and filter out irrelevant data points over extended periods. Conversely, CNNs might be preferred for their robustness against noise and their effectiveness in extracting high-level spatial features from high-dimensional data.

The selection between these models is therefore not just about their theoretical capabilities but also about practical considerations like computational resources, data specificity, and the specific requirements of the EEG analysis task. For a comprehensive approach that balances between spatial and temporal data challenges in EEG, hybrid models that integrate the spatial pattern recognition capability of CNNs with the temporal processing strength of LSTMs are often the most effective solution, as discussed in the reviews by Yu et al. (2022).

7. Conclusion

This project aims to compare how LSTM and GRU models detect emotions in EEG signals to ascertain if comparative accuracy of LSTM is achieved with the computationally cheaper model GRU.

Both models performed well and as expected the LSTM model performed better than the GRU model. In comparing the two models, it is clear that the LSTM model offers superior accuracy, making it the preferred choice for applications requiring precise emotion classification from EEG signals. However, the GRU model's performance is noteworthy, especially considering its reduced computational requirements. This makes the GRU model a practical alternative in scenarios where computational resources are limited.

In this project the GRU model has proved to be slightly inferior to the LSTM model in terms of accuracy as shown in Table 1, however the Gru model performed well enough to be a real alternative to LSTM. According to a quote attributed to Albert Einstein:

"A model should be as simple as possible, but no simpler than that" − Albert Einstein.

As such a model which offers more simplicity with a small cost to accuracy is a trade-off which is defensible.

Some shortcomings are lack of repeated experiments with several data sets to see how the accuracy and other scores in table 1 match up with different EEG data. In the information regarding the EEG dataset "Emotions.csv" there is no mention of participants reporting on the actual emotion invoked by a clip. If an assumption is made regarding the emotional state of the participant based on what clip he or she saw there could be mislabelled data giving inaccurate results.

The GRU model might, however, not be the simplest and cheapest method. Future projects can look into the possibility of conducting even computationally cheaper classifications of emotions in EEG.

8. References

1. J. Bird, J., Ekart, A., D. Buckingham, C., & R. Faria, D. (2019). Mental Emotional Sentiment Classification with an EEG-based Brain-machine Interface. *Conference: The International Conference on Digital Image & Signal*.

2. Koelstra, S., Mühl, C., Soleymani, M., Lee, J. S., Yazdani, A., Ebrahimi, T., Pun, T., Nijholt, A., & Patras, I. (2012). DEAP: A database for emotion analysis; Using physiological signals. *IEEE Transactions on Affective Computing*, *3*(1), 18–31. https://doi.org/10.1109/T-AFFC.2011.15

3. Xu, G., Ren, T., Chen, Y., & Che, W. (2020). A one-dimensional CNN-LSTM model for epileptic seizure recognition using EEG signal analysis. Frontiers in Neuroscience Link

4. Najafi, T., Jaafar, R., Remli, R., & Wan Zaidi, W. A. (2022). A classification model of EEG signals based on rnn-lstm for diagnosing focal and generalized epilepsy. Sensors Link

5. Garcia-Moreno, F. M., Bermudez-Edo, M., & others. (2020). CNN-LSTM deep Learning classifier for motor imagery EEG detection using a low-invasive and low-cost BCI headband. IEEE Access Link

6. Jiang, H., Jiao, R., Wang, Z., Zhang, T., & Wu, L. (2021). Construction and analysis of emotion computing model based on LSTM. Complexity, 2021. Link

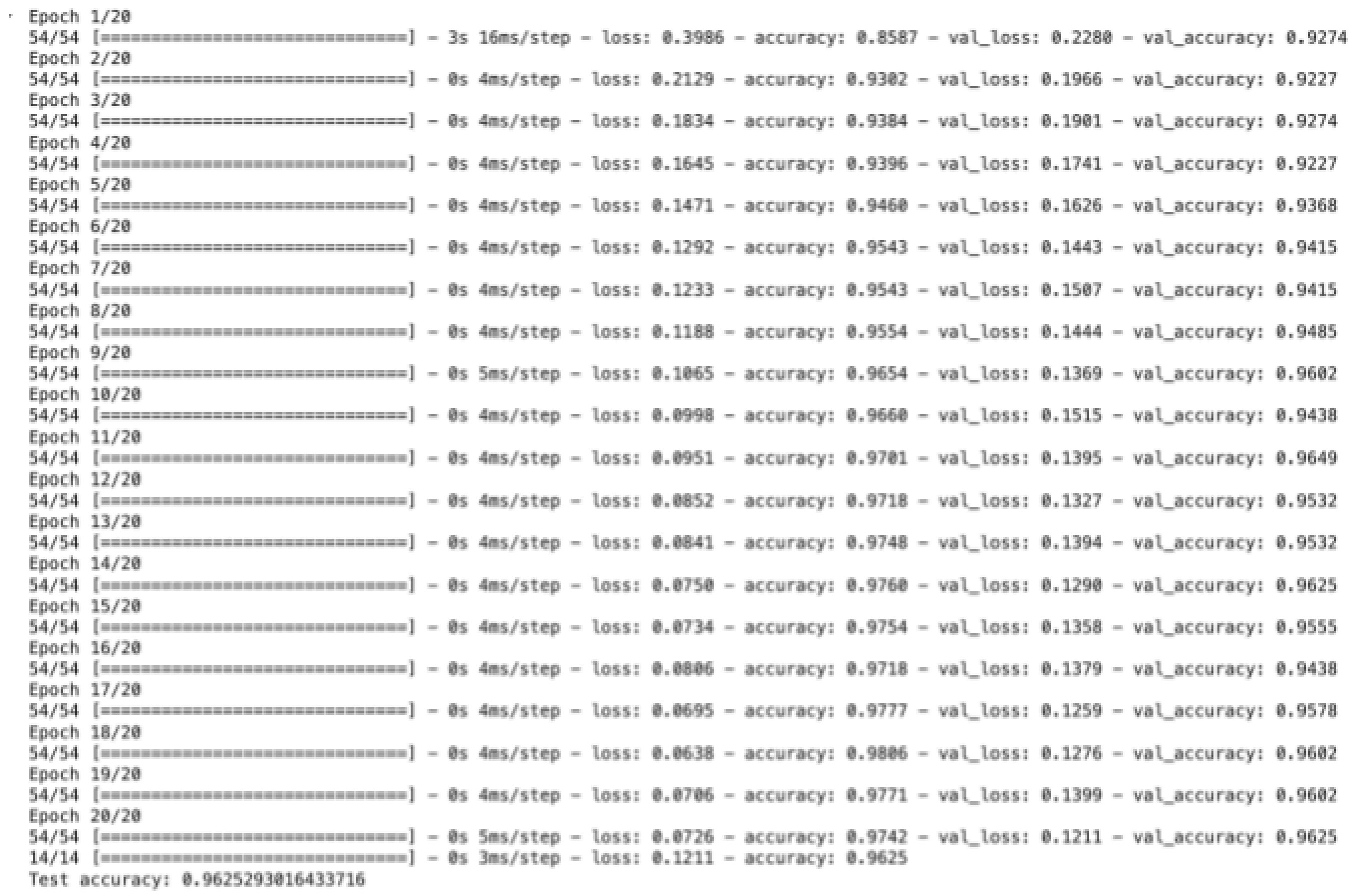
7. Yu, J., de Antonio, A., & Villalba-Mora, E. (2022). Deep learning (CNN, RNN) applications for smart homes: A systematic review. Computers, 11(2). MDPI. Link

8. Rim, B., Sung, N.J., Min, S., & Hong, M. (2020). Deep learning in physiological signal data: A survey. Sensors, 20(4), 969. MDPI. Link

9. DatabaseCamp. (n.d.). Long Short-Term Memory (LSTM). Retrieved from https://databasecamp.de/en/ml/lstms (20.05.24)

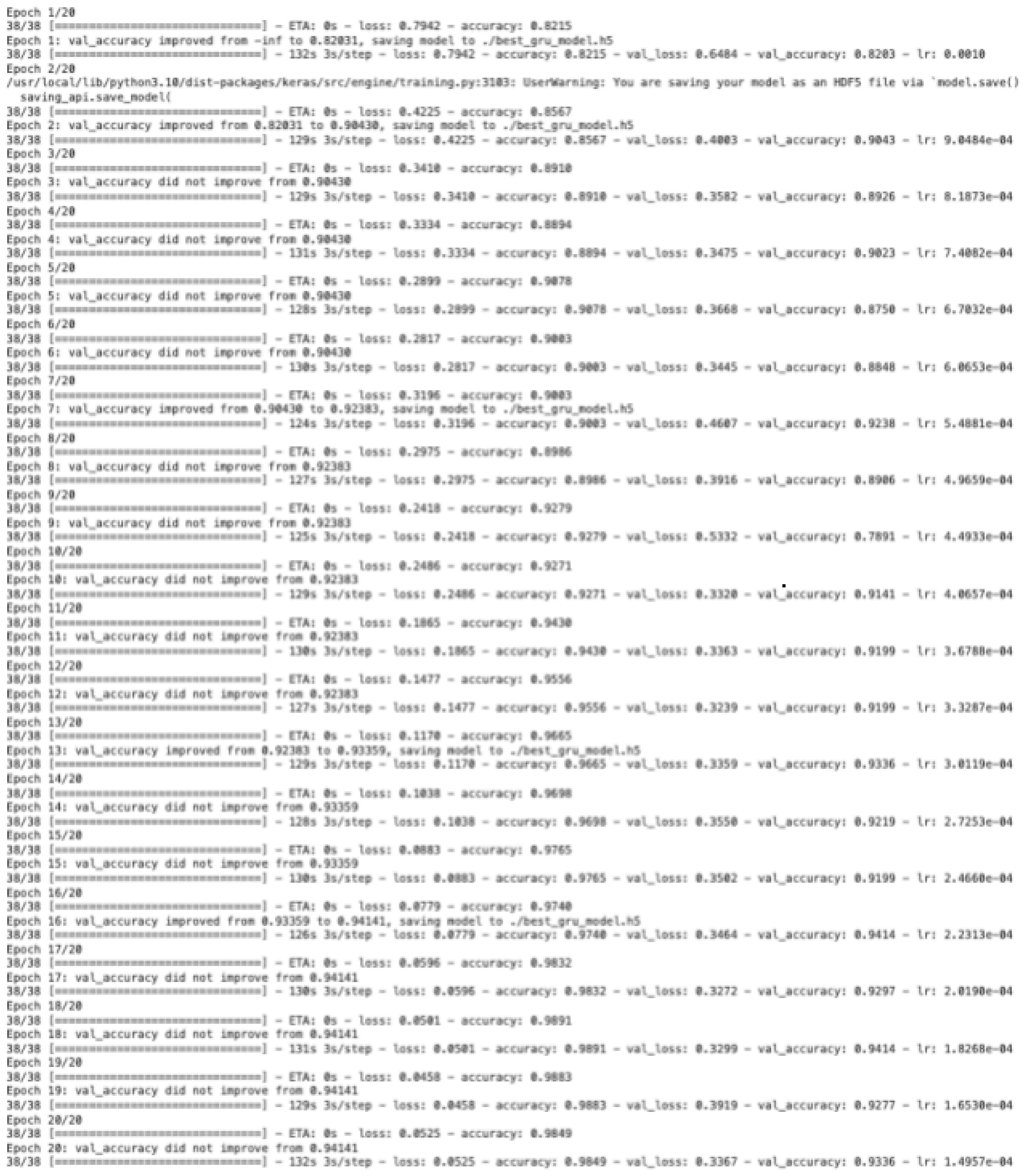
10. Github project https://github.com/LarsStorholt/EEGEmotionClassification

Appendix



```
Epoch 1/20
54/54 [==============================] - 3s 16ms/step - loss: 0.3986 - accuracy: 0.8587 - val_loss: 0.2280 - val_accuracy: 0.9274
Epoch 2/20
54/54 [==============================] - 0s 4ms/step - loss: 0.2129 - accuracy: 0.9302 - val_loss: 0.1966 - val_accuracy: 0.9227
Epoch 3/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1834 - accuracy: 0.9384 - val_loss: 0.1901 - val_accuracy: 0.9274
Epoch 4/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1645 - accuracy: 0.9396 - val_loss: 0.1741 - val_accuracy: 0.9227
Epoch 5/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1471 - accuracy: 0.9460 - val_loss: 0.1626 - val_accuracy: 0.9368
Epoch 6/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1292 - accuracy: 0.9543 - val_loss: 0.1443 - val_accuracy: 0.9415
Epoch 7/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1233 - accuracy: 0.9543 - val_loss: 0.1507 - val_accuracy: 0.9415
Epoch 8/20
54/54 [==============================] - 0s 4ms/step - loss: 0.1188 - accuracy: 0.9554 - val_loss: 0.1444 - val_accuracy: 0.9485
Epoch 9/20
54/54 [==============================] - 0s 5ms/step - loss: 0.1065 - accuracy: 0.9654 - val_loss: 0.1369 - val_accuracy: 0.9602
Epoch 10/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0998 - accuracy: 0.9660 - val_loss: 0.1515 - val_accuracy: 0.9438
Epoch 11/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0951 - accuracy: 0.9701 - val_loss: 0.1395 - val_accuracy: 0.9649
Epoch 12/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0852 - accuracy: 0.9718 - val_loss: 0.1327 - val_accuracy: 0.9532
Epoch 13/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0841 - accuracy: 0.9748 - val_loss: 0.1394 - val_accuracy: 0.9532
Epoch 14/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0750 - accuracy: 0.9760 - val_loss: 0.1290 - val_accuracy: 0.9625
Epoch 15/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0734 - accuracy: 0.9754 - val_loss: 0.1358 - val_accuracy: 0.9555
Epoch 16/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0806 - accuracy: 0.9718 - val_loss: 0.1379 - val_accuracy: 0.9438
Epoch 17/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0695 - accuracy: 0.9777 - val_loss: 0.1259 - val_accuracy: 0.9578
Epoch 18/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0638 - accuracy: 0.9806 - val_loss: 0.1276 - val_accuracy: 0.9602
Epoch 19/20
54/54 [==============================] - 0s 4ms/step - loss: 0.0706 - accuracy: 0.9771 - val_loss: 0.1399 - val_accuracy: 0.9602
Epoch 20/20
54/54 [==============================] - 0s 5ms/step - loss: 0.0726 - accuracy: 0.9742 - val_loss: 0.1211 - val_accuracy: 0.9625
14/14 [==============================] - 0s 3ms/step - loss: 0.1211 - accuracy: 0.9625
Test accuracy: 0.9625293016433716
```

Fig 7: Accuracy test of our LSTM model shows a 96.2% accuracy rate of identyfying emotions.



```
Epoch 1/20
38/38 [==============================] - ETA: 0s - loss: 0.7942 - accuracy: 0.8215
Epoch 1: val_accuracy improved from -inf to 0.82031, saving model to ./best_gru_model.h5
38/38 [==============================] - 132s 3s/step - loss: 0.7942 - accuracy: 0.8215 - val_loss: 0.6484 - val_accuracy: 0.8203 - lr: 0.0010
Epoch 2/20
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()
  saving_api.save_model(
38/38 [==============================] - ETA: 0s - loss: 0.4225 - accuracy: 0.8567
Epoch 2: val_accuracy improved from 0.82031 to 0.90430, saving model to ./best_gru_model.h5
38/38 [==============================] - 129s 3s/step - loss: 0.4225 - accuracy: 0.8567 - val_loss: 0.4003 - val_accuracy: 0.9043 - lr: 9.0484e-04
Epoch 3/20
38/38 [==============================] - ETA: 0s - loss: 0.3410 - accuracy: 0.8910
Epoch 3: val_accuracy did not improve from 0.90430
38/38 [==============================] - 129s 3s/step - loss: 0.3410 - accuracy: 0.8910 - val_loss: 0.3582 - val_accuracy: 0.8926 - lr: 8.1873e-04
Epoch 4/20
38/38 [==============================] - ETA: 0s - loss: 0.3334 - accuracy: 0.8894
Epoch 4: val_accuracy did not improve from 0.90430
38/38 [==============================] - 131s 3s/step - loss: 0.3334 - accuracy: 0.8894 - val_loss: 0.3475 - val_accuracy: 0.9023 - lr: 7.4082e-04
Epoch 5/20
38/38 [==============================] - ETA: 0s - loss: 0.2899 - accuracy: 0.9078
Epoch 5: val_accuracy did not improve from 0.90430
38/38 [==============================] - 128s 3s/step - loss: 0.2899 - accuracy: 0.9078 - val_loss: 0.3668 - val_accuracy: 0.8750 - lr: 6.7032e-04
Epoch 6/20
38/38 [==============================] - ETA: 0s - loss: 0.2817 - accuracy: 0.9003
Epoch 6: val_accuracy did not improve from 0.90430
38/38 [==============================] - 130s 3s/step - loss: 0.2817 - accuracy: 0.9003 - val_loss: 0.3445 - val_accuracy: 0.8848 - lr: 6.0653e-04
Epoch 7/20
38/38 [==============================] - ETA: 0s - loss: 0.3196 - accuracy: 0.9003
Epoch 7: val_accuracy improved from 0.90430 to 0.92383, saving model to ./best_gru_model.h5
38/38 [==============================] - 124s 3s/step - loss: 0.3196 - accuracy: 0.9003 - val_loss: 0.4607 - val_accuracy: 0.9238 - lr: 5.4881e-04
Epoch 8/20
38/38 [==============================] - ETA: 0s - loss: 0.2975 - accuracy: 0.8986
Epoch 8: val_accuracy did not improve from 0.92383
38/38 [==============================] - 127s 3s/step - loss: 0.2975 - accuracy: 0.8986 - val_loss: 0.3916 - val_accuracy: 0.8906 - lr: 4.9659e-04
Epoch 9/20
38/38 [==============================] - ETA: 0s - loss: 0.2418 - accuracy: 0.9279
Epoch 9: val_accuracy did not improve from 0.92383
38/38 [==============================] - 125s 3s/step - loss: 0.2418 - accuracy: 0.9279 - val_loss: 0.5332 - val_accuracy: 0.7891 - lr: 4.4933e-04
Epoch 10/20
38/38 [==============================] - ETA: 0s - loss: 0.2486 - accuracy: 0.9271
Epoch 10: val_accuracy did not improve from 0.92383
38/38 [==============================] - 129s 3s/step - loss: 0.2486 - accuracy: 0.9271 - val_loss: 0.3320 - val_accuracy: 0.9141 - lr: 4.0657e-04
Epoch 11/20
38/38 [==============================] - ETA: 0s - loss: 0.1865 - accuracy: 0.9430
Epoch 11: val_accuracy did not improve from 0.92383
38/38 [==============================] - 130s 3s/step - loss: 0.1865 - accuracy: 0.9430 - val_loss: 0.3363 - val_accuracy: 0.9199 - lr: 3.6788e-04
Epoch 12/20
38/38 [==============================] - ETA: 0s - loss: 0.1477 - accuracy: 0.9556
Epoch 12: val_accuracy did not improve from 0.92383
38/38 [==============================] - 127s 3s/step - loss: 0.1477 - accuracy: 0.9556 - val_loss: 0.3239 - val_accuracy: 0.9199 - lr: 3.3287e-04
Epoch 13/20
38/38 [==============================] - ETA: 0s - loss: 0.1170 - accuracy: 0.9665
Epoch 13: val_accuracy improved from 0.92383 to 0.93359, saving model to ./best_gru_model.h5
38/38 [==============================] - 129s 3s/step - loss: 0.1170 - accuracy: 0.9665 - val_loss: 0.3359 - val_accuracy: 0.9336 - lr: 3.0119e-04
Epoch 14/20
38/38 [==============================] - ETA: 0s - loss: 0.1038 - accuracy: 0.9698
Epoch 14: val_accuracy did not improve from 0.93359
38/38 [==============================] - 128s 3s/step - loss: 0.1038 - accuracy: 0.9698 - val_loss: 0.3550 - val_accuracy: 0.9219 - lr: 2.7253e-04
Epoch 15/20
38/38 [==============================] - ETA: 0s - loss: 0.0883 - accuracy: 0.9765
Epoch 15: val_accuracy did not improve from 0.93359
38/38 [==============================] - 130s 3s/step - loss: 0.0883 - accuracy: 0.9765 - val_loss: 0.3502 - val_accuracy: 0.9199 - lr: 2.4660e-04
Epoch 16/20
38/38 [==============================] - ETA: 0s - loss: 0.0779 - accuracy: 0.9740
Epoch 16: val_accuracy improved from 0.93359 to 0.94141, saving model to ./best_gru_model.h5
38/38 [==============================] - 126s 3s/step - loss: 0.0779 - accuracy: 0.9740 - val_loss: 0.3464 - val_accuracy: 0.9414 - lr: 2.2313e-04
Epoch 17/20
38/38 [==============================] - ETA: 0s - loss: 0.0596 - accuracy: 0.9832
Epoch 17: val_accuracy did not improve from 0.94141
38/38 [==============================] - 130s 3s/step - loss: 0.0596 - accuracy: 0.9832 - val_loss: 0.3272 - val_accuracy: 0.9297 - lr: 2.0190e-04
Epoch 18/20
38/38 [==============================] - ETA: 0s - loss: 0.0501 - accuracy: 0.9891
Epoch 18: val_accuracy did not improve from 0.94141
38/38 [==============================] - 131s 3s/step - loss: 0.0501 - accuracy: 0.9891 - val_loss: 0.3299 - val_accuracy: 0.9414 - lr: 1.8268e-04
Epoch 19/20
38/38 [==============================] - ETA: 0s - loss: 0.0458 - accuracy: 0.9883
Epoch 19: val_accuracy did not improve from 0.94141
38/38 [==============================] - 129s 3s/step - loss: 0.0458 - accuracy: 0.9883 - val_loss: 0.3919 - val_accuracy: 0.9277 - lr: 1.6530e-04
Epoch 20/20
38/38 [==============================] - ETA: 0s - loss: 0.0525 - accuracy: 0.9849
Epoch 20: val_accuracy did not improve from 0.94141
38/38 [==============================] - 132s 3s/step - loss: 0.0525 - accuracy: 0.9849 - val_loss: 0.3367 - val_accuracy: 0.9336 - lr: 1.4957e-04
```

Fig 8: Accuracy test of our GRU model shows a 94.1% accuracy rate of identyfying emotions.