

Human Activity Recognition (HAR) using WISDM

Group 15

Project Final Report
Machine Learning
CAP 5610

Group Members:

Thiwanka Dissanayaka (Coordinator): sahan@ucf.edu

Shahd Alnofaie: sh467442@ucf.edu

Chathura Keshan: chathura.keshan@ucf.edu

Sandamini Senaratne: lo602443@ucf.edu

Zack Willis: john.willis@ucf.edu

Contents

1	Problem Definition	3
2	Data Set	3
3	Pipeline	4
4	Explanatory Data Analysis	4
5	Methodology	6
5.1	Multilayer Perceptron (MLP)	7
5.1.1	Architecture	7
5.1.2	Backpropagation and Optimization	7
5.1.3	MLP Implementation	8
5.2	Convolutional Neural Network (CNN)	8
5.2.1	Key Components	8
5.2.2	CNN Implementation	9
5.3	Recurrent Neural Network (RNN)	9
5.3.1	Mathematical Formulation	9
5.3.2	RNN Implementation	10
5.4	Bidirectional Long Short-Term Memory (BiLSTM)	10
5.4.1	LSTM Architecture	10
5.4.2	Bidirectional Extension	11
5.4.3	Bi-LSTM Implementation	11
5.5	Temporal Convolutional Network (TCN)	11
5.5.1	Model Architecture	12
5.5.2	Decoder:	12

5.5.3	TCN Implementation	12
6	Evaluation Metrics	13
6.1	Accuracy	13
6.2	Precision	13
6.3	Recall (Sensitivity or True Positive Rate)	13
6.4	F1-Score	14
7	Hyperparameters Tuning	14
8	Results	15
8.1	MLP	15
8.2	CNN	16
8.3	RNN	18
8.4	BiLSTM	19
8.5	TCN	21
8.6	Overall Results	22
9	Discussion and Conclusion	23

1 Problem Definition

In recent years, human activity recognition (HAR) has gained considerable attention in both industry and academia due to the widespread use of sensors, particularly accelerometers and gyroscopes, in smartphones and smartwatches. HAR finds applications in various fields, especially for monitoring an individual's lifestyle and functional capabilities. This project focuses on identifying common human activities from sensor data collected by smartphones and watches using deep learning techniques.

2 Data Set

This project uses the well-known WISDM dataset, collected by the Department of Computer and Information Science at Fordham University. Data was gathered from the accelerometer and gyroscope sensors in the smartphones and smartwatches of 51 people as they performed 18 diverse activities of daily living. Moreover, each person performed each activity for 3 minutes. The dataset records X, Y, and Z axis values from each sensor (e.g., smartphone accelerometer, smartwatch gyroscope) every 50ms for all activities. Furthermore, the data set consists of 15,630,426 raw measurements together with the corresponding activity.

Activity	Code	Class Index
Walking	A	0
Jogging	B	1
Stairs	C	2
Sitting	D	3
Standing	E	4
Typing	F	5
Brushing Teeth	G	6
Eating Soup	H	7
Eating Chips	I	8
Eating Pasta	J	9
Drinking from Cup	K	10
Eating Sandwich	L	11
Kicking (Soccer Ball)	M	12
Playing Catch w/Tennis Ball	O	13
Dribbling (Basketball)	P	14
Writing	Q	15
Clapping	R	16
Folding Clothes	S	17

Table 1: The 18 activities represented in the dataset along with their class indices.

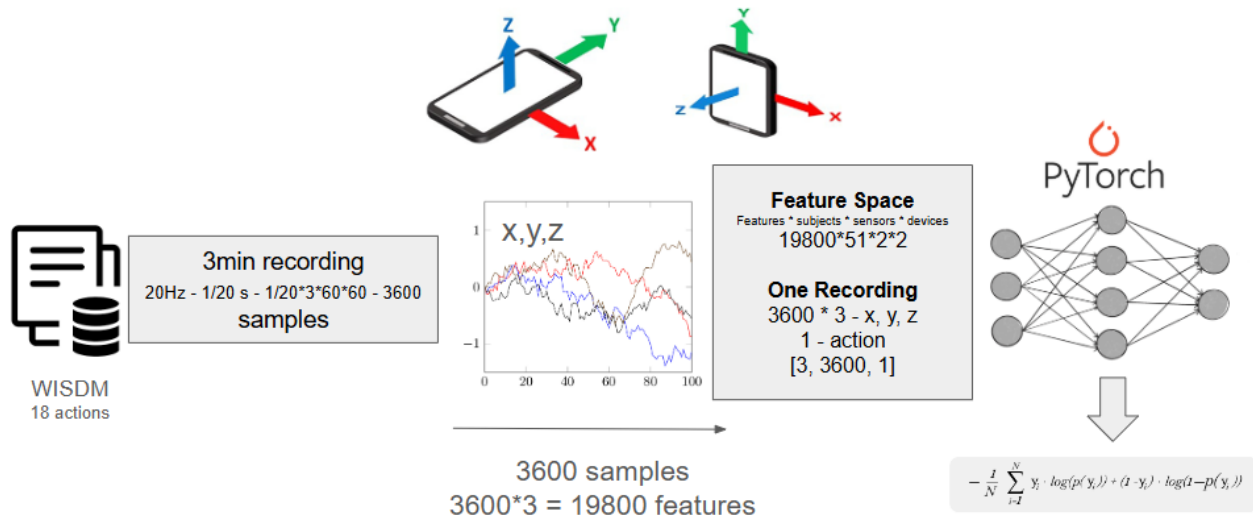


Figure 1: Pipeline

3 Pipeline

Figure 1 illustrates the implementation pipeline. As described in Section 2, each activity was recorded for a duration of 3 minutes at a sampling frequency of 20 Hz. Consequently, each activity comprises 3,600 samples (per x , y and z axes) for each sensor, recorded for each individual. For the implementation, the data was structured into sequences with 19,800 features per sample, effectively reducing the dimensionality of the feature space to 4,039,200.

4 Explanatory Data Analysis

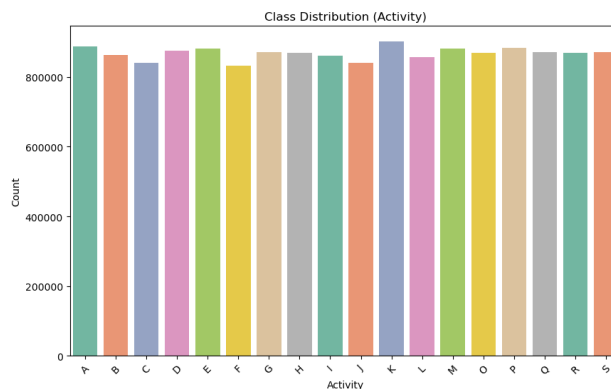


Figure 2: Class Distribution

The figure 2 illustrates the distribution of the 18 activity classes in the dataset. Each class has a nearly equal number of samples, indicating that the dataset is well-balanced. This ensures that the model performs fairly across all activity categories.

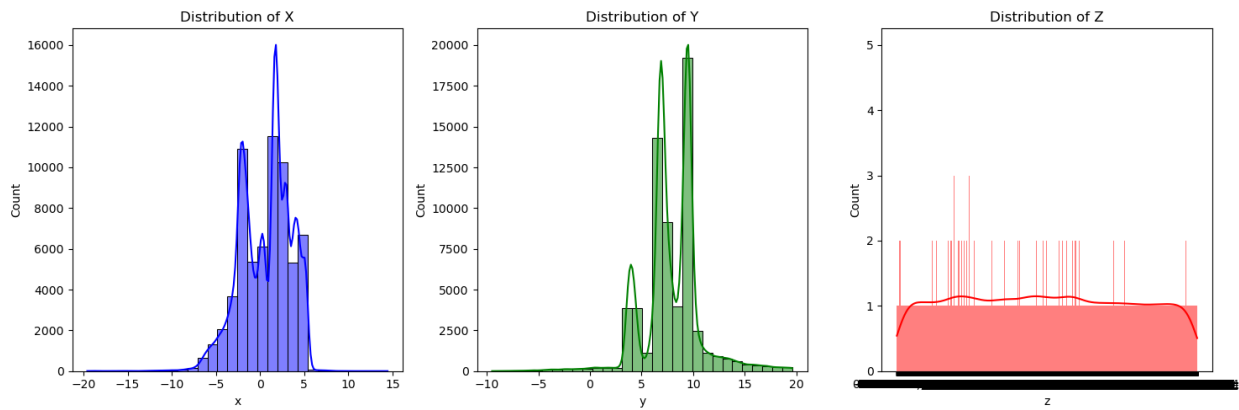


Figure 3: Distribution of X, Y, and Z variables

The figure 3 consists of three histograms, each representing the distribution of sensor data along the x, y, and z axes.

The blue histogram shows the frequency of different values recorded for the x-axis data. The distribution appears to have multiple peaks, indicating a multi-modal distribution. Most values range between -10 and 10, with a significant concentration near 0.

The green histogram displays the frequency of values recorded for the y-axis data. Like the x-axis, the y-axis data shows a multi-modal distribution with distinct peaks. The majority of the values range between 0 and 10, with fewer occurrences in the negative range.

The red histogram represents the z-axis data. The distribution for the z-axis appears flatter compared to the other two axis. Therefore, it can be said that the values in z-axis is approximately uniformly distributed.

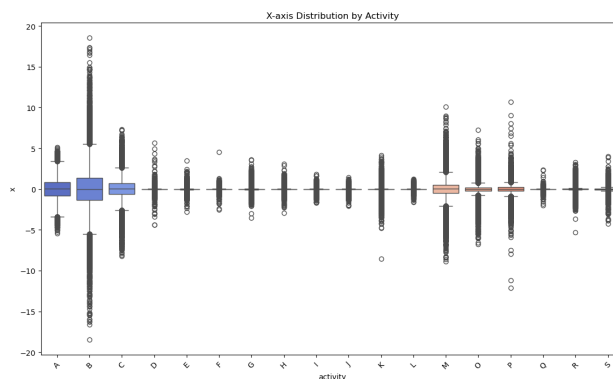


Figure 4: X variable distribution with activity

Activities such as A, B, and M show a wider spread in x-axis values compared to others, indicating more variability in the x-axis acceleration during these activities. Other activities, such as E or H, show narrower distributions, suggesting more consistent sensor readings for those activities.

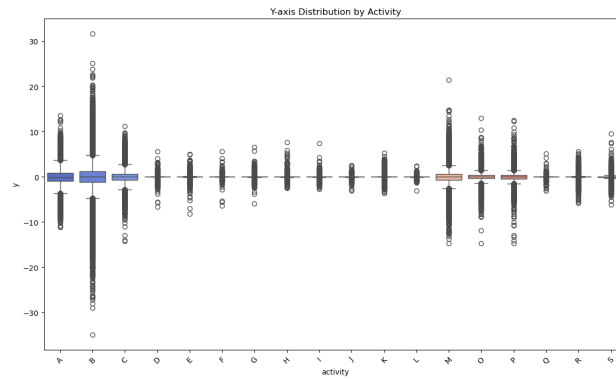


Figure 5: Y variable distribution with activity

Activities like A and B show a wider spread, indicating higher variability in y-axis readings, likely from dynamic movements during these activities. Activities like E, H, and O exhibit narrower boxplots, suggesting more consistent movements along the y-axis.

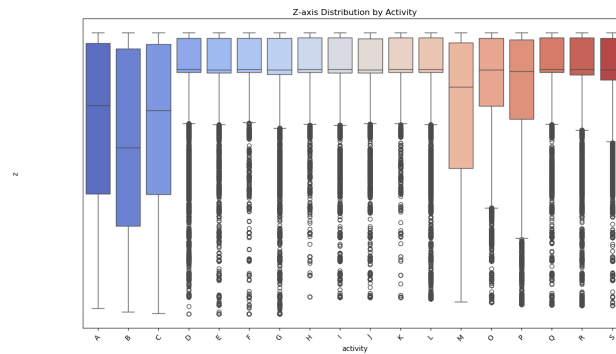


Figure 6: Z variable distribution with activity

Activities A, B, and M show a much larger spread, indicating more variability in z-axis sensor data. This is evident from the taller boxes and longer whiskers for these activities. Activities such as E, H, and K have narrower boxes, suggesting less variation in z-axis readings, which may correspond to more consistent movements.

5 Methodology

Human action recognition can be identified as a classification problem in machine learning. Particularly, X, Y, and Z axis values gathered from the sensors are used to determine the correct activity, and this is acquired using ML models for classification. Therefore, Multi-Layer Perceptrons, Convolutional Neural Networks, Recurrent Neural Networks, Long-Short-Term Memory, and Temporal Convolutional Networks are implemented for the data in this project.

5.1 Multilayer Perceptron (MLP)

The Multilayer Perceptron (MLP) [6] is a type of feedforward artificial neural network that maps input features to target outputs by optimizing weights through backpropagation. MLPs are extensively used for solving complex classification and regression problems, leveraging their ability to approximate any continuous function.

5.1.1 Architecture

The MLP consists of multiple layers of neurons:

- An **input layer**, where each neuron corresponds to a feature in the input data.
- One or more **hidden layers** that learn non-linear transformations of the data.
- An **output layer**, with neurons corresponding to the target classes in multi-class classification.

Each neuron performs a weighted sum of its inputs, applies a bias, and passes the result through a non-linear activation function. The mathematical formulation is as follows:

$$z_j^{[l]} = \sum_{i=1}^{n_{l-1}} W_{ij}^{[l]} a_i^{[l-1]} + b_j^{[l]},$$

$$a_j^{[l]} = f(z_j^{[l]}),$$

where, $z_j^{[l]}$: Input to neuron j in layer l , $W_{ij}^{[l]}$: Weight connecting neuron i in layer $l-1$ to neuron j in layer l , $b_j^{[l]}$: Bias of neuron j in layer l , $a_i^{[l-1]}$: Activation of neuron i in layer $l-1$, $f(\cdot)$: Activation function.

5.1.2 Backpropagation and Optimization

Training an MLP involves minimizing the loss function using backpropagation. The gradients of the loss with respect to weights and biases are computed as:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{[l]}} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b_j^{[l]}}.$$

The weights are updated iteratively using optimization algorithms like Stochastic Gradient Descent (SGD) or Adam:

$$W_{ij}^{[l]} \leftarrow W_{ij}^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W_{ij}^{[l]}},$$

$$b_j^{[l]} \leftarrow b_j^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial b_j^{[l]}},$$

where η is the learning rate.

5.1.3 MLP Implementation

The MLP model was implemented with four hidden layers containing 256, 128, 64 and 32 neurons, respectively. The ReLU activation function was used for each layer, and the Adam optimizer was employed for training with a learning rate of 0.001. The loss function was calculated using cross-entropy loss. The model was trained for 50 epochs with a batch size of 32.

5.2 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) [3, 5] is a specialized type of neural network designed for pattern recognition and data structured in grid-like topology, such as images. CNNs are widely used in computer vision tasks, including image classification, object detection, and segmentation.

5.2.1 Key Components

CNNs consist of three primary types of layers: convolutional layers, pooling layers, and fully connected layers. These layers are stacked sequentially to extract hierarchical features from the input data.

Convolutional Layer

The convolutional layer performs feature extraction by applying a set of learnable filters (kernels) to the input data. Each filter computes a weighted sum over a local region of the input, producing an activation map. The operation is defined as:

$$z_{i,j}^{[l]} = \sum_{m=1}^k \sum_{n=1}^k W_{m,n}^{[l]} x_{i+m,j+n} + b^{[l]},$$

where, $z_{i,j}^{[l]}$: Activation at position (i, j) in the output feature map of layer l , $W_{m,n}^{[l]}$: Filter weights of size $k \times k$, $x_{i+m,j+n}$: Input at position $(i + m, j + n)$, $b^{[l]}$: Bias term for the filter.

Each convolutional layer uses an activation function (e.g., ReLU) to introduce non-linearity:

$$a_{i,j}^{[l]} = \text{ReLU}(z_{i,j}^{[l]}) = \max(0, z_{i,j}^{[l]}).$$

Pooling Layer

The pooling layer reduces the spatial dimensions of the feature maps while retaining essential features. The most common pooling operation is max-pooling:

$$a_{i,j}^{[l]} = \max_{p=1}^s \max_{q=1}^s z_{i+p,j+q}^{[l]},$$

where s is the pooling size.

Pooling reduces the computational complexity and helps control overfitting by providing translational invariance.

Fully Connected Layer

Fully connected layers map the extracted features from the convolutional and pooling layers to the output classes. Each neuron in the fully connected layer is connected to every neuron in the previous layer:

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]},$$

where, $z^{[l]}$: Linear combination of inputs at layer l , $W^{[l]}$: Weight matrix, $b^{[l]}$: Bias vector, $a^{[l-1]}$: Activation from the previous layer.

The final fully connected layer uses the softmax function for multi-class classification:

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j=1}^C \exp(z_j)},$$

where C is the number of classes.

5.2.2 CNN Implementation

The CNN model was implemented using the optimal hyperparameters for the activity recognition task. The network consists of 3 convolutional layers with 64 filters in the first layer, 128 filters in the second layer, and 256 filters in the third layer. Each convolutional layer uses a kernel size of 3×3 with a padding of 1 to preserve spatial dimensions. To introduce non-linearity, the ReLU activation function was applied to all layers.

The training process was optimized using the Adam optimizer with a learning rate of 0.001, ensuring efficient weight updates. Additionally, a learning rate scheduler was employed to decay the learning rate periodically to improve convergence. These hyperparameters were carefully tuned to balance performance and computational efficiency, enabling the model to capture complex spatiotemporal features effectively for activity recognition.

5.3 Recurrent Neural Network (RNN)

The Recurrent Neural Network (RNN) [7, 8] is a neural network architecture designed to process sequential data by maintaining a hidden state that evolves over time. Unlike feedforward networks, RNNs have cycles in their architecture, enabling them to retain information about previous inputs, making them well-suited for tasks such as time-series prediction, language modeling, and sequential classification.

5.3.1 Mathematical Formulation

An RNN operates on a sequence of inputs $X = \{x_1, x_2, \dots, x_T\}$, where $x_t \in \mathbb{R}^d$ represents the input at time step t . The network maintains a hidden state h_t and computes it iteratively using

the following equations:

$$h_t = \phi_h(W_{xh}x_t + W_{hh}h_{t-1} + b_h),$$

$$o_t = \phi_o(W_{ho}h_t + b_o),$$

where, $h_t \in \mathbb{R}^h$: Hidden state at time t , $W_{xh} \in \mathbb{R}^{d \times h}$: Input-to-hidden weight matrix, $W_{hh} \in \mathbb{R}^{h \times h}$: Hidden-to-hidden weight matrix, $W_{ho} \in \mathbb{R}^{h \times o}$: Hidden-to-output weight matrix, $b_h \in \mathbb{R}^h$ and $b_o \in \mathbb{R}^o$: Bias vectors, ϕ_h : Activation function for the hidden layer (commonly tanh or ReLU), ϕ_o : Activation function for the output layer.

5.3.2 RNN Implementation

The final setup for the RNN is 6 layers with 64 nodes each. Various combinations of 3, 6, and 9 layers with 32, 64, and 128 nodes were used. These either showed no improvement or worse results compared to the final configuration. The learning rate is set to 1e-4, with larger rates resulting in oscillation. Batch size did not seem to have any impact when set to 32, 64, or 128.

5.4 Bidirectional Long Short-Term Memory (BiLSTM)

The Bidirectional Long Short-Term Memory (BiLSTM) [1] network is an advanced recurrent neural network architecture designed to learn from sequential data by capturing dependencies in both forward and backward directions. It builds upon the Long Short-Term Memory (LSTM) architecture, which effectively addresses the limitations of traditional recurrent neural networks (RNNs), such as vanishing and exploding gradients.

5.4.1 LSTM Architecture

An LSTM consists of memory cells and three gates:

1. Forget Gate: Determines which information to discard from the cell state.
2. Input Gate: Decides what new information to store in the cell state.
3. Output Gate: Controls the output based on the cell state.

The operations in an LSTM cell at time t are:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f),$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i),$$

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o), \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$

where, x_t : Input vector at time t , h_{t-1} : Previous hidden state, c_t : Cell state, f_t, i_t, o_t : Forget, input, and output gates, respectively, W_f, W_i, W_c, W_o : Weight matrices, b_f, b_i, b_c, b_o : Bias terms, σ : Sigmoid activation function, \tanh : Hyperbolic tangent activation function, \odot : Element-wise multiplication.

5.4.2 Bidirectional Extension

The BiLSTM extends the LSTM by processing the input sequence in both forward and backward directions. For a sequence of length T :

$$\begin{aligned}
\vec{h}_t &= \text{LSTM}_{\text{forward}}(x_1, \dots, x_t), \\
\overleftarrow{h}_t &= \text{LSTM}_{\text{backward}}(x_T, \dots, x_t).
\end{aligned}$$

The final hidden state combines the forward and backward states:

$$h_t = [\vec{h}_t; \overleftarrow{h}_t],$$

where $[\cdot; \cdot]$ denotes concatenation.

The combined hidden state h_t is passed to the output layer, typically with a softmax activation function for multi-class classification:

$$\hat{y}_t = \text{softmax}(W_y h_t + b_y),$$

where, W_y : Output weight matrix, b_y : Bias vector.

5.4.3 Bi-LSTM Implementation

xyz sequence of each sample was fed into a bi-directional LSTM model consisting of two LSTM layers. The size of the hidden layer is 64. RELU was used as the activation function while Adam was the activation function. The loss was calculated from cross-entropy loss. The model was trained with 50 epochs and batch size 32. The used learning rate is 1e-3.

5.5 Temporal Convolutional Network (TCN)

The Temporal Convolutional Network (TCN) [2] is a convolutional architecture designed for sequential data tasks such as multi-class classification. It leverages hierarchical temporal feature extraction through stacked convolutional layers and pooling mechanisms. Unlike recurrent models, TCNs process sequences using convolutions, which capture long-range temporal dependencies in an efficient manner.

5.5.1 Model Architecture

A TCN operates on sequences of input data $X_t \in \mathbb{R}^{F_0}$, where $t = 1, \dots, T$ denotes the time steps, and F_0 is the feature dimension. The true label for each time step t is $y_t \in \{1, \dots, C\}$, where C is the number of classes.

Encoder: The encoder consists of L convolutional layers. Each layer applies 1D convolutions with filters parameterized as $W^{(l)} \in \mathbb{R}^{F_l \times d \times F_{l-1}}$, where F_l is the number of filters at layer l , d is the kernel size, and F_{l-1} is the number of input channels.

$$\hat{E}_i^{(l)}(t) = f \left(b_i^{(l)} + \sum_{t'=1}^d \langle W_i^{(l)}(t'), E^{(l-1)}(t + d - t') \rangle \right),$$

where $f(\cdot)$ is the activation function.

Pooling: After each convolutional layer, max-pooling with a width of 2 is applied along the time axis:

$$T_l = \frac{T_{l-1}}{2}.$$

Pooling reduces the temporal resolution, allowing the model to capture long-term patterns efficiently.

Normalization: Channel-wise normalization is applied to the pooled activations. Given the maximum activation $m = \max_i \hat{E}_i^{(l)}(t)$ and a small constant ϵ , the normalized activation is:

$$E^{(l)}(t) = \frac{\hat{E}^{(l)}(t)}{m + \epsilon}.$$

5.5.2 Decoder:

The decoder mirrors the encoder, using upsampling instead of pooling. Upsampling is performed by repeating each entry in the time dimension. The operations are ordered as upsample, convolve, and normalize.

The final layer predicts the probabilities of each class for each time step using a softmax activation function:

$$\hat{Y}_t = \text{softmax}(UE^{(1)}(t) + c),$$

where $U \in \mathbb{R}^{C \times F_1}$ is the weight matrix, and $c \in \mathbb{R}^C$ is the bias vector.

5.5.3 TCN Implementation

The model design incorporated carefully selected hyperparameters to optimize performance for activity recognition. The kernel size was initialized between 2 and 5 to effectively capture the temporal granularity of actions. To ensure broad temporal coverage while maintaining computational efficiency, the dilation factor was increased exponentially, following a sequence of 1, 2, 4, and so

on, across layers configured with 16, 32, 64, and 128 filters. A dropout rate of 0.2 was applied to mitigate overfitting, addressing the challenges posed by the dense nature of activity recognition data. The optimal learning rate was set at 1e-3. The Adam optimizer was employed for its adaptability, and ReLU was used as the activation function to introduce non-linearity and enhance model learning. This structured design ensured both robustness and scalability. The model was trained for 100 epochs.

6 Evaluation Metrics

To assess the performance of the machine learning model, we employ several evaluation metrics, each providing insights into different aspects of the model's classification ability. These metrics are especially relevant for multi-class classification tasks.

6.1 Accuracy

Accuracy measures the proportion of correctly predicted samples out of the total number of samples. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}}$$

While accuracy is a straightforward metric, it can be misleading in cases where the dataset is imbalanced. If one class dominates the dataset, a model could achieve high accuracy by predominantly predicting the majority class, even if it performs poorly on the minority classes.

6.2 Precision

Precision evaluates the proportion of true positive predictions among all positive predictions. It is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Precision is particularly useful in scenarios where false positives are costly.

6.3 Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of actual positive samples that are correctly predicted by the model. It is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

6.4 F1-Score

The F1-Score provides a harmonic mean of precision and recall, balancing the trade-off between the two metrics. It is given by:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-Score is particularly valuable when the dataset is imbalanced, as it penalizes extreme disparities between precision and recall.

7 Hyperparameters Tuning

Hyperparameter tuning was performed systematically for all models to optimize their performance. A range of hyperparameters, such as the number of channels, kernel size, dropout rate, and learning rate, was explored using a grid search approach with cross-validation. This process involved defining a parameter grid for each model and evaluating various combinations of these parameters to identify the configuration that yielded the best accuracy. The tuning process employed GridSearchCV, which executed an exhaustive search over the parameter space with a fixed number of cross-validation folds to ensure robustness and generalizability of the results. The best-performing parameters for each model were selected based on the highest cross-validated accuracy, and the outcomes provided valuable insights into how each model responded to different hyperparameter configurations. This comprehensive tuning ensured that all models were optimized for their respective tasks.

8 Results

8.1 MLP

Class	Precision	Recall	F1-Score	Support
0	0.68	0.74	0.71	1366
1	0.89	0.81	0.85	1358
2	0.44	0.50	0.47	1323
3	0.33	0.40	0.36	1338
4	0.36	0.39	0.38	1337
5	0.33	0.44	0.38	1280
6	0.54	0.41	0.47	1323
7	0.35	0.41	0.38	1294
8	0.36	0.30	0.33	1306
9	0.31	0.31	0.31	1221
10	0.39	0.35	0.37	1337
11	0.32	0.17	0.22	1344
12	0.43	0.51	0.47	1330
13	0.45	0.55	0.48	1321
14	0.54	0.45	0.49	1301
15	0.38	0.37	0.38	1351
16	0.54	0.46	0.50	1228
17	0.40	0.37	0.38	1072

Table 2: Classification Report for MLP

The MLP model achieved varying levels of precision, recall, and F1-scores across different classes. Some classes, such as 1 and 0, exhibited high precision and recall values, indicating strong performance. Other classes, like 9 and 11, demonstrated lower metrics, suggesting difficulty in distinguishing those activities.

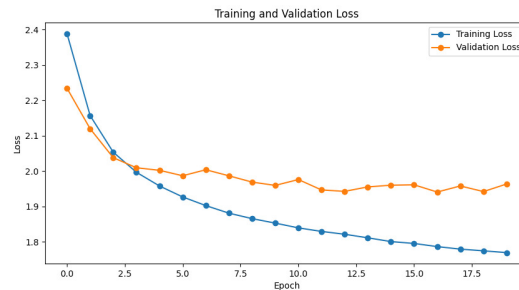


Figure 7: MLP Loss Plot

The training loss decreased steadily, indicating that the model effectively learned during training. Validation loss showed a similar trend but started to slightly diverge toward the end, which could indicate minor overfitting.

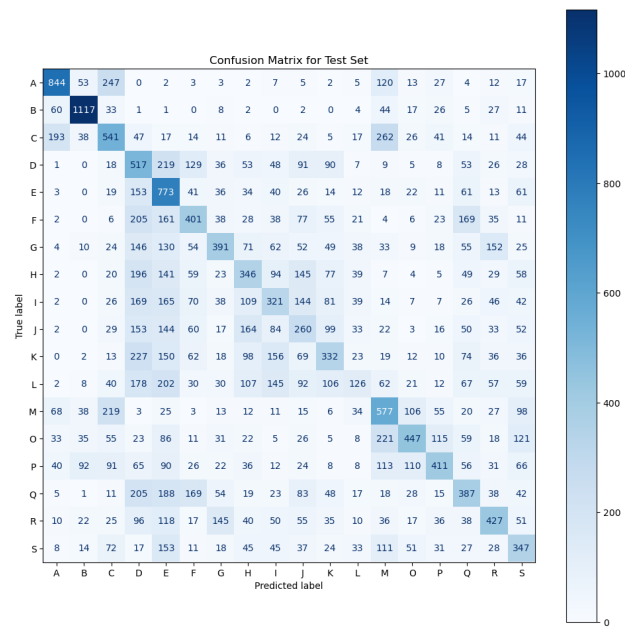


Figure 8: MLP Confusion Matrix

The confusion matrix highlights that the model performed well for certain classes (A and B), with higher diagonal values. Misclassifications are evident in some classes with significant off-diagonal values, suggesting overlapping features between activities.

8.2 CNN

Class	Precision	Recall	F1-Score	Support
0	0.90	0.89	0.89	1382
1	0.93	0.95	0.94	1323
2	0.71	0.76	0.74	1289
3	0.31	0.62	0.42	1267
4	0.42	0.66	0.51	1322
5	0.79	0.59	0.68	1329
6	0.74	0.72	0.73	1234
7	0.62	0.50	0.56	1269
8	0.56	0.47	0.51	1333
9	0.65	0.47	0.54	1308
10	0.54	0.47	0.50	1368
11	0.52	0.33	0.41	1314
12	0.63	0.66	0.65	1316
13	0.67	0.69	0.68	1316
14	0.79	0.73	0.76	1343
15	0.76	0.63	0.69	1338
16	0.80	0.74	0.77	1272
17	0.63	0.70	0.66	1107

Table 3: Classification Report for CNN

The model’s precision, recall, and F1-scores vary across classes. While some classes (e.g., Class 0 and Class 1) show high performance with F1-scores above 0.8, others (e.g., Class 3 and Class 11) have significantly lower F1-scores, indicating difficulty in distinguishing these classes. This suggests class imbalance or overlapping features between certain classes.

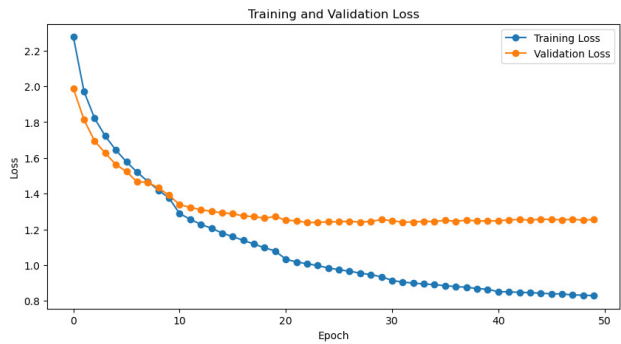


Figure 9: CNN Loss Plot

The training and validation loss decrease steadily over the epochs, indicating convergence of the model. However, the gap between training and validation loss towards the end suggests potential overfitting.

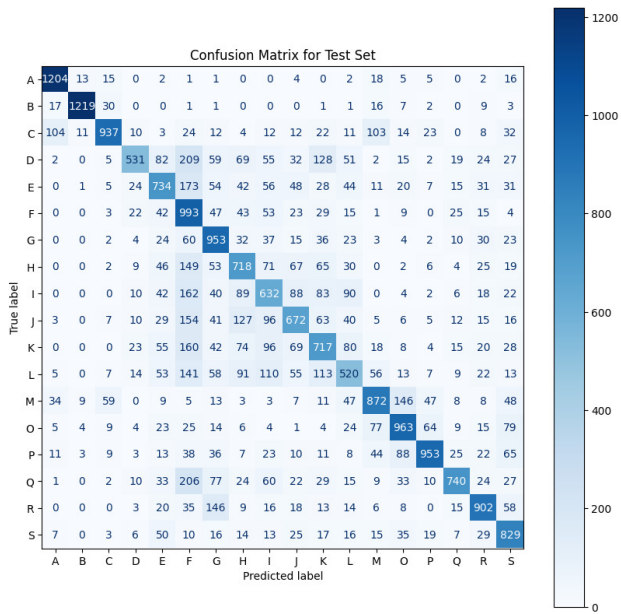


Figure 10: CNN Confusion Matrix

The confusion matrix highlights the model’s prediction performance across all classes. Diagonal entries indicate correct predictions, with some classes showing high accuracy (e.g., Classes A and B). Off-diagonal entries reveal misclassifications, particularly among similar classes.

8.3 RNN

Class	Precision	Recall	F1-Score	Support
0	0.30	0.31	0.30	13711
1	0.37	0.41	0.39	13308
2	0.29	0.17	0.22	13426
3	0.66	0.76	0.71	13764
4	0.59	0.73	0.65	13946
5	0.65	0.77	0.71	13451
6	0.48	0.58	0.53	13735
7	0.46	0.56	0.51	13646
8	0.55	0.45	0.50	13613
9	0.52	0.49	0.51	13632
10	0.55	0.50	0.53	13951
11	0.50	0.47	0.49	13682
12	0.27	0.15	0.19	13794
13	0.25	0.21	0.23	13348
14	0.33	0.26	0.29	13764
15	0.60	0.82	0.69	13810
16	0.48	0.51	0.49	13846
17	0.29	0.30	0.29	13840

Table 4: Classification Report for RNN

The RNN model demonstrates varied performance across classes, with some classes (e.g., Class 3 and Class 5) achieving higher F1-scores (above 0.7), while others (e.g., Classes 2, 12, and 13) show significantly lower F1-scores, indicating poor model performance for these categories. This reflects the model's struggle with distinguishing certain classes, likely due to feature overlap or insufficient representation.

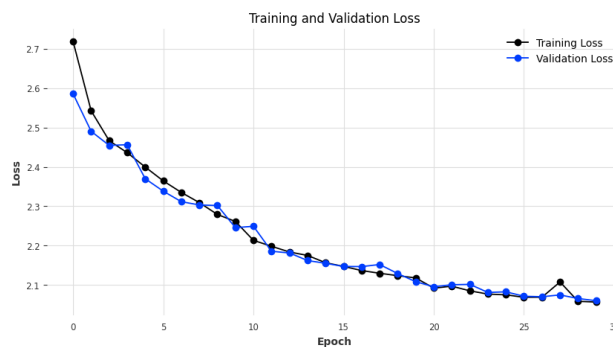


Figure 11: RNN Loss Plot

The training and validation loss decrease steadily over the epochs, indicating that the model is learning effectively. The convergence of the two loss curves toward the end suggests minimal overfitting and a reasonably well-trained model.

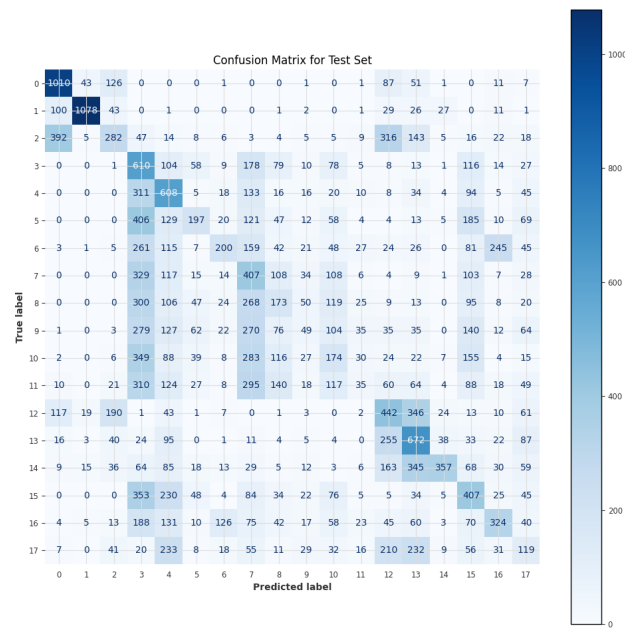


Figure 12: RNN Confusion Matrix

The confusion matrix reveals the RNN's classification accuracy for each class. While certain classes (e.g., Classes 3 and 5) have high correct predictions (diagonal entries), many off-diagonal entries highlight significant misclassifications, particularly among classes with similar patterns.

8.4 BiLSTM

Class	Precision	Recall	F1-Score	Support
0	0.88	0.84	0.86	1366
1	0.91	0.94	0.92	1358
2	0.75	0.69	0.72	1323
3	0.47	0.59	0.52	1338
4	0.58	0.56	0.57	1337
5	0.64	0.64	0.64	1280
6	0.72	0.71	0.72	1323
7	0.61	0.57	0.59	1294
8	0.48	0.47	0.47	1306
9	0.36	0.58	0.44	1221
10	0.59	0.47	0.52	1337
11	0.47	0.40	0.43	1344
12	0.63	0.62	0.63	1330
13	0.69	0.65	0.67	1321
14	0.72	0.70	0.71	1301
15	0.67	0.65	0.66	1351
16	0.81	0.67	0.73	1228
17	0.61	0.64	0.62	1072

Table 5: Classification Report for BiLSTM

The BiLSTM model shows varying performance across classes. High-performing classes include Class 0 and Class 1, with F1-scores of 0.86 and 0.92, respectively. However, certain classes (e.g., Class 9 and Class 11) have relatively lower F1-scores (0.44 and 0.43, respectively), indicating challenges in correctly identifying these categories.

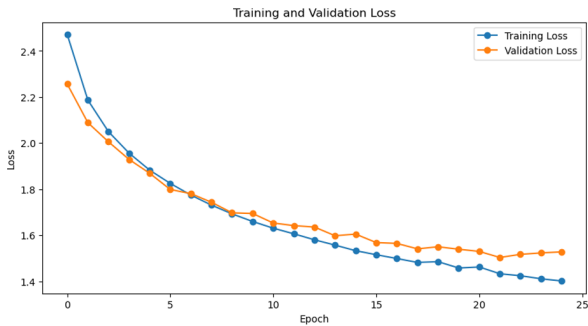


Figure 13: BiLSTM Loss Plot

The training and validation loss exhibit a steady decline over the epochs, demonstrating effective model learning. The gap between training and validation losses remains small, indicating good generalization and minimal overfitting.

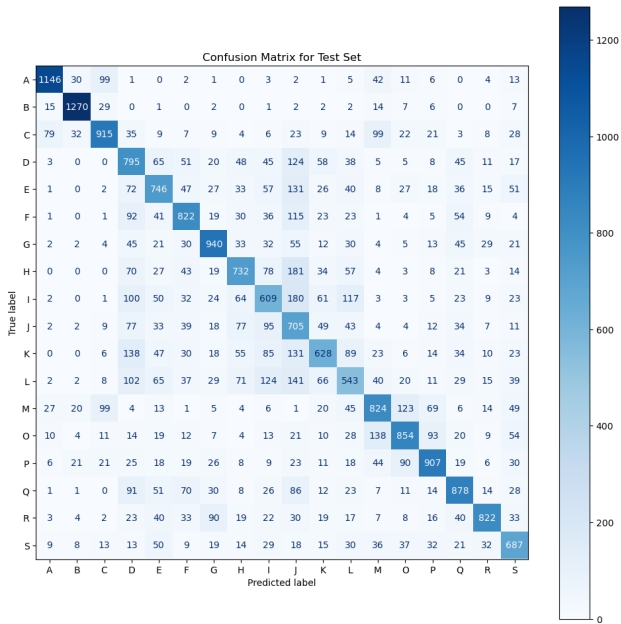


Figure 14: BiLSTM Confusion Matrix

The confusion matrix highlights the model’s classification performance. Diagonal elements, especially for Classes A and B, indicate high accuracy. However, some off-diagonal misclassifications suggest overlapping features or insufficient data separation for certain classes, such as Classes J and K.

8.5 TCN

Class	Precision	Recall	F1-Score	Support
0	0.86	0.93	0.90	1288
1	0.97	0.93	0.95	1307
2	0.86	0.70	0.77	1342
3	0.78	0.40	0.53	1312
4	0.58	0.55	0.57	1324
5	0.39	0.75	0.51	1324
6	0.57	0.76	0.65	1258
7	0.53	0.57	0.55	1266
8	0.47	0.49	0.48	1288
9	0.58	0.52	0.54	1301
10	0.52	0.51	0.52	1409
11	0.50	0.40	0.45	1287
12	0.69	0.66	0.67	1329
13	0.70	0.72	0.71	1330
14	0.82	0.70	0.75	1369
15	0.81	0.56	0.66	1322
16	0.74	0.71	0.73	1263
17	0.62	0.75	0.68	1111

Table 6: Classification Report for TCN

The TCN model demonstrates high performance for certain classes, such as Class 1 (F1-score: 0.95) and Class 0 (F1-score: 0.90). However, other classes, such as Class 3 (F1-score: 0.53) and Class 11 (F1-score: 0.45), exhibit lower scores, indicating challenges in distinguishing these categories. The results suggest that while the TCN performs well overall, certain classes require additional attention.



Figure 15: TCN Loss Plot

The training and validation loss steadily decrease over the epochs, with the validation loss plateauing at later epochs. The minimal gap between training and validation loss indicates good generalization with minimal overfitting.

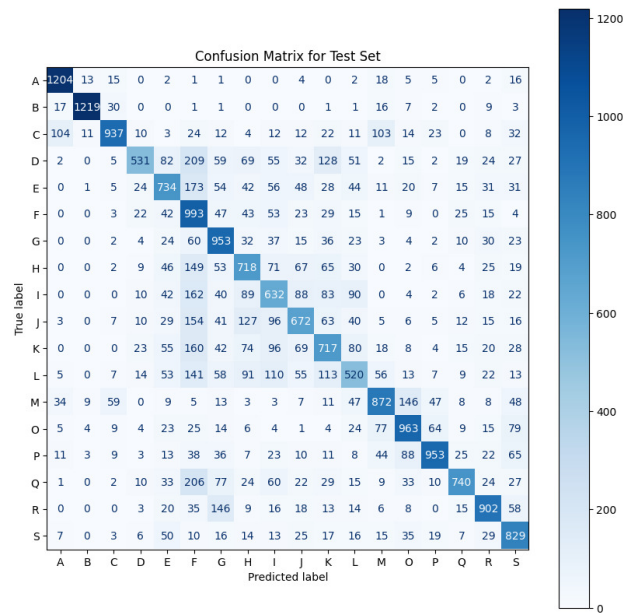


Figure 16: TCN Confusion Matrix

In classification matrix, Diagonal entries particularly for Classes A and B, indicate strong accuracy. However, off-diagonal misclassifications, such as between Classes J and K, highlight areas where the model struggles with distinguishing similar patterns.

8.6 Overall Results

Models	Precision	Recall	F1-Score	Accuracy
MLP	0.45	0.44	0.44	0.4443
CNN	0.67	0.64	0.65	0.6432
RNN	0.45	0.47	0.46	0.4703
BiLSTM	0.64	0.63	0.64	0.6327
TCN	0.67	0.64	0.65	0.6440

Table 7: Evaluation metrics across the models

The evaluation metrics in Table 7 highlight the performance differences among the tested models. The MLP model demonstrates the weakest performance, with the lowest precision (0.45), recall (0.44), F_1 -score (0.44), and accuracy (44.43%), indicating its limited capability to handle sequential data. RNN improves slightly, achieving an accuracy of 47.03%, but its F_1 -score (0.46) remains low, likely due to challenges in learning long-term dependencies. BiLSTM shows a marked improvement with a precision of 0.64, recall of 0.63, and accuracy of 63.27%, leveraging its bidirectional architecture to capture temporal dependencies effectively. CNN and TCN models perform similarly, with TCN achieving the highest accuracy (64.40%) and an F_1 -score of 0.65, showcasing its ability to model long-range dependencies efficiently through dilated convolutions. Overall, the

results emphasize the superiority of advanced architectures like TCN and BiLSTM over simpler models for activity recognition tasks.

9 Discussion and Conclusion

Project findings show varying performance of MLP, CNN, RNN, BiLSTM, and TCN models for human activity recognition. Most effective models were TCN and BiLSTM, with the highest performance metrics. TCN excels at representing temporal dependencies using dilated convolutions, making it ideal for the sequential dataset. BiLSTM's robust performance has been attributed to its bidirectional processing of input sequences. Simpler designs like MLP and RNN were unable to handle complicated temporal and spatial patterns in data, resulting in worse results.

LSTM and TCN models work well, however overlapping activities like "eating soup" and "eating pasta" dramatically lower model performance. These activities have similar x, y, and z axis sensor data, leading to misunderstanding during training and testing. The models struggle to distinguish between classes due to the absence of unique spatiotemporal patterns in the data. This shows that even advanced systems struggle to differentiate between occupations with identical motion characteristics. To address this difficulty, new feature engineering strategies like data aggregation or hybrid architectures are needed to better capture small variations in overlapping operations.

Even though Transformer-based models are popular for sequence modeling, they nevertheless struggle with overlapping activities, as shown in previous studies [4]. Misclassifications were caused by high overlap in features in some activities in this study.

Future project work may overcome problems with overlapping activities and improve model performance. Combining Temporal Convolutional Networks (TCN) with Long Short-Term Memory networks (LSTM) can improve temporal modeling and identify comparable behaviors like "eating soup" and "eating pasta."

Additional elements, such as gyroscope readings or environmental information, can differentiate overlapping actions. Improved feature engineering, attention mechanisms, and data augmentation strategies can enhance the model's ability to detect minor changes in activity patterns. Exploring Transformer-based hybrid models may provide ways to balance their potential with other architectures, despite their known issues with overlapping activities. These developments could considerably improve the reliability and accuracy of human activity identification systems.

Contributions

- Thiwanka Dissanayaka: TCN implementation, code aggregation and structuring, and leading the project
- Shahd Alnofaie: CNN implementation, presentation preparation
- Chathura Keshan: BiLSTM implementation, report writing
- Poorna Sandamini Senaratne: MLP implementation, report writing
- Zack Willis: RNN implementation, presentation preparation

References

- [1] A. Graves and J. Schmidhuber. “Framewise phoneme classification with bidirectional LSTM networks”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*. Vol. 4. 2005, 2047–2052 vol. 4. DOI: 10.1109/IJCNN.2005.1556215.
- [2] Colin Lea et al. *Temporal Convolutional Networks: A Unified Approach to Action Segmentation*. 2016. arXiv: 1608.08242 [cs.CV]. URL: <https://arxiv.org/abs/1608.08242>.
- [3] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.
- [4] Shuangjian Li et al. *P2LHAP: Wearable sensor-based human activity recognition, segmentation and forecast through Patch-to-Label Seq2Seq Transformer*. 2024. arXiv: 2403.08214 [cs.CV]. URL: <https://arxiv.org/abs/2403.08214>.
- [5] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: <https://arxiv.org/abs/1511.08458>.
- [6] Marius-Constantin Popescu et al. “Multilayer perceptron and neural networks”. In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009).
- [7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536. URL: <https://api.semanticscholar.org/CorpusID:205001834>.
- [8] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG]. URL: <https://arxiv.org/abs/1912.05911>.