Benchmarking Convolutional and Recurrent Deep Learning Networks for Human Activity Recognition

by

Philip Sigillito
Department of Computer Science
College of Information Science and Technology
University of Nebraska at Omaha
Omaha, NE 68182-0500
psigillito@unomaha.edu

For

23SP CSCI8910, Master of Science Capstone

Spring, 2023

Wednesday, May 3, 2023

Abstract

Human activity recognition, HAR, is a pattern recognition problem where the goal is to identify a person's actions based on observed input. These observations generally come in the form of either videos, which are composed of two-dimensional flat images, or sensors such as gyroscopes, accelerometers, and magnetometers. Traditional statistical approaches such as SVM machines, decision trees, bayes nets, and k-nearest neighbor clustering have long been applied to the problem. In recent years, these have been replaced by more robust deep learning approaches such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). This is largely because deep learning techniques can outperform traditional approaches and do not require extensive domain knowledge or data preparation to be accurate. Many deep learning networks that combine the properties of different node types have been researched for increasing performance. A common approach is to combine a CNN layer with an RNN layer. The CNN layers extract features in the input while the RNN extracts the sequential relationship of inputs. Surveyed literature proposes models that combine CNNs and different types of RNNs but do not provide a comprehensive comparison benchmarking performance of these models. Additionally, the surveyed literature often limits the metrics of performance and only focuses on accuracy. This paper addresses that lack of comparison, and provides a benchmarking of the commonly used deep learning architectures discussed in the surveyed literature. It provides more subtlety to the performance comparison and shows that selecting the 'best' architecture is dependent on what metrics are most valued. We see that CNN prepended networks generally perform better than networks without CNN layers, the CNN-GRU model performs best when the input size is small, CNN-BILSTM and CNN-LSTM networks perform best when the input size is large, and the CNN and CNN-CNN networks perform well against the training data but degrade more significantly than other networks during validation although they train significantly faster than recurrent networks.

Table of Contents

I. Introduction

I.1 Problem:

Human Activity Recognition, HAR, is a popular topic with a wide range of applications such as health tracking, human-robot interaction, surveillance, elder care, and moderating online content. HAR solutions were once dominated by traditional statistical approaches such as SVMs, decision trees, and clustering algorithms but these approaches have largely been replaced by deep learning. There are many types of input available for HAR but they generally fall into two categories: sequential two dimensional images (video) or sequential sensor data. Input contains a spatial dimension to understanding what is being conveyed in each step of the input and also contains a temporal dimension of understanding how each step in the input relates to each other.

As deep learning approaches have grown in popularity, a wide variety of architectures are being applied to the problem. One popular approach is to combine different layer types to leverage their unique strengths. For example, combining a CNN layer with an LSTM layer is increasingly popular. The survey of literature for this project suggests that most research is now motivated to increase the accuracy and variety of classifications as well as reduce the computational cost of models to better run on constrained devices such as smartphones and watches.

I.2 Motivations:

When selecting a deep learning architecture to use for Human Activity Recognition, it is valuable to have a comprehensive comparison of how each model will perform. Most papers surveyed state a CNN-LSTM model is ideal but they do not fully quantify this assertion. For example, none of the papers surveyed include networks that use RNN or GRU layers in their comparisons. [Perez-Gamboa et. al. 2021 & Mutegeki and Han 2020] compare CNNs, LSTMs, and CNN-LSTM networks with various depths but do not discuss other types of RNNs. [Moradi et. al. 2022] also compare CNN, LSTM, CNN-LSTM and BILSTMs but does not include CNN-BILSTM networks. This paper aims to provide a comprehensive comparison to quantify performance differences. This paper includes a benchmarking of architectures that are commonly mentioned in the surveyed literature:

- Traditional RNN
- CNN
- GRU
- LSTM
- BILSTM
- CNN-RNN
- CNN-GRU
- CNN-LSTM
- CNN-BILSTM

Aspects of the network architectures such as layer size, depth, and extra layers such as batch normalization and dropout are kept as consistent as possible between implementations. The networks are tested against two sensor datasets: WISDM and MHEALTH. The datasets use different sampling frequencies and sensors. After benchmarking this paper briefly looks at tuning the best performing model to see the impact that selecting the best hyperparameters can impose

on a network's performance.

The papers surveyed for this project were also inconsistent in the metrics used to measure performance. To address the lack of comprehensive measurements in previous papers: accuracy, recall, precision, and training time are recorded. Confusion matrices for interesting networks are also included.

I.3 Significance:

This paper implements several deep learning models to classify actions using the WISDM and MHEALTH sensor data set. This paper compares the performance of these models using accuracy, precision, recall, and training time. This project also applies hyperparameter tuning to the models to measure how the tuning improves the models and provide insight into which features are most impactful to performance. This paper addresses that lack of comprehensive comparison benchmark and provides more subtlety to the performance comparison by showing that selecting the 'best' architecture is dependent on factors such as input size, noise of the data, and which metric is most valued.

I.4 Challenges

Although there is a dearth of research applying deep learning to HAR, there is not a lot of information for thoroughly comparing how those networks compare in performance. A meaningful comparison needs to include network architectures that use different node types (RNN, CNN, etc.) while keeping other aspects of the networks constant such as layer count, dropout, data trained on, and layer size. Because node types are inherently different, creating a fair benchmarking is challenging. For example, CNN networks perform differently based on their kernel size but RNN networks have no concept of a kernel.

I.5 Objectives

The objective of this paper is to provide a comprehensive comparison of deep learning models for HAR. Networks that simply represent CNN and RNN networks as well as networks that combine the two node types are tested. Additionally, Keras Tuner is applied to one of the best performing networks, CNN-GRU, to briefly explore how tuning the hyperparameters can impact performance. Accuracy, precision, recall, and training time are included for each network architecture. Graphs of the training and confusion matrices are included in the Appendix.

II. Overview

II.1. History of the problem

Human Action recognition, HAR, has existed for decades. This is in large part because of the vast array of practical applications. Traditional statistical and machine learning approaches were the main form of HAR for many years. These approaches provide good accuracy for sensor data but, as shown in [Mutegeki and Han 2020], require a lot of domain knowledge and data prepping.

Support Vector Machines, SVMs, are a common technique for classification and multi-classification. SVMs work by finding a hyperplane that divides a set of data into two classes. The hyperplane is applicable to a problem space with any number of dimensions. For an

N-dimensional space problem, the hyperplane is an N-1 subspace that divides it [Chelliah 2023]. The name ‘support vector’ machine is based on the closest points to the hyperplane in each class which ‘support’ the location of the plane. If these data points are moved or removed, the hyperplane will also move. In contrast, if the other data points are altered, the hyperplane will remain unchanged [Chelliah 2023]. Since HAR is a form of classification problem, SVMs can be applied to the problem.

Decision trees and random forests are two more long-standing techniques for solving classification problems. [Qian et al. 2010] utilize decision trees where each node is an SVM. Decision trees work by dividing the input by a condition that is either true or false at each node. The input will then continue down the branches of the tree, choosing their path based on whether they satisfy each node’s condition until they reach a leaf. Techniques such as XGBost and AdaBoost extend decision trees for more performant predictions. For example, AdaBoost creates many trees and uses a weighted voting system to accept the best value.

Another type of technique is clustering such as K-means clustering. K-means clustering is an unsupervised approach where unlabelled data is grouped into clusters. The developer decides the K number of clusters desired and assigns each datapoint to a cluster. After this setup the algorithm iteratively assigns each point to the cluster center that is closest. This is done iteratively, each time calculating the cluster centers and assigning entries to the closest cluster. This continues until data points are no longer switching clusters. This is an interesting approach because it does not require understanding the meaning of the datasets.

II.2. State of the art

[Moradi et. al. 2022] provide a general survey of some of the different state of the art techniques common today. They include simpler approaches such as SVM machines as well as more advanced deep learning techniques such as CNN, LSTM, BILSTM, and CNN-LSTM networks. This survey found that BILSTM and CNN-LSTM approaches were the most accurate although they took considerably longer to train.

Novel approaches using non-deep learning are still being researched. For example, [Qian et al. 2010] utilize SVMs and a data tree structure to make HAR classifications. As cited in [Moradi et. al. 2022] and [Mutegeki and Han 2020], these novel approaches are declining in popularity because deep learning is more robust and can be more widely and easily applied. The cited reason for this is because it removes much of the data prepping and domain expertise of the things being classified to be accurate. For example, a CNN layer automatically identifies features during training and stacking CNN layers can result in finding more complex features.

Many types of deep learning networks are being applied as state of the art solutions to human activity recognition such as CNN, RNN, LSTM, and BILSTM networks. As described in [Mutegeki and Han 2020], the CNN-LSTM model is appealing because it leverages the spatially deep nature of the CNN to extract features with the strength of the LSTM’s temporal understanding. The state of the art models that perform best are also likely to be tuned for the problem they are solving. For example, [Joshi et. al. 2021] trained a CNN-LSTM network and then used the Keras Tuner to improve its network’s architecture. State of the art technologies such as Keras Tuner have made tuning networks and understanding how the architecture is

modified more accessible. Values such as the number of layers, layer depth, number of epochs, learning rate, and loss function can be tuned to make a more optimal model.

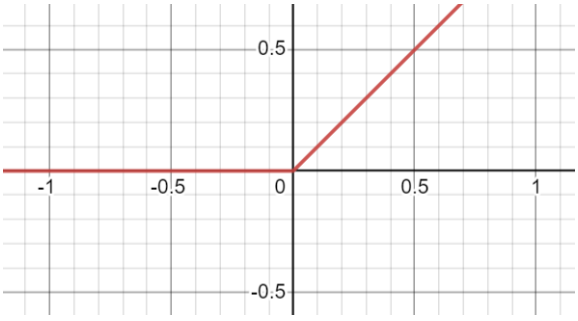
III. Techniques

III.1. Principles, Concepts, and Theoretical Foundations of the research prob

This paper is an extension of previous works that measure how deep learning models perform for human activity recognition. Most papers measuring performance do so in the context of asserting their novel architecture. To provide a comprehensive comparison, the following architectures are included in this report:

- Traditional RNN
- CNN
- GRU
- LSTM
- BILSTM
- CNN-RNN
- CNN-GRU
- CNN-LSTM
- CNN-BILSTM

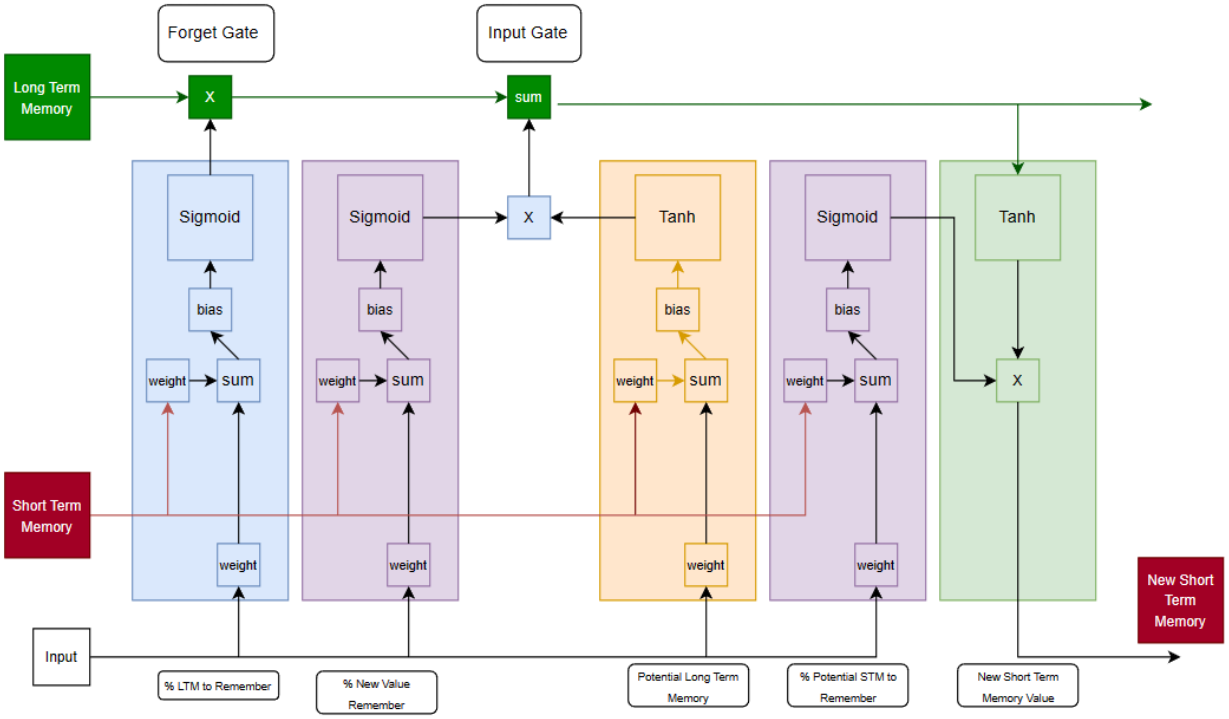
Each of these architectures have their own strengths and weaknesses. Convolutional neural networks, CNNs, are useful for feature extraction as they can find patterns in input. When CNNs are stacked, more complicated patterns can be found. These are commonly used in image processing as it uses filters that allow it to better understand the context of an image in comparison to its predecessor the MultiLayer Perceptron, MLP. Whereas MLPs take in a single tensor, the CNN's filter allows it to learn feature maps in the image. CNNs typically utilize the Rectified Linear Activation, ReLU, activation function. ReLU is simple to calculate and less susceptible to the vanishing gradient problem than other activation functions such as sigmoid and tanh as outlined in [Brownlee 2021]. The equation for ReLU is: $y = max(0, x)$. This captures the value of the input when it is positive but treats all negative values as zero. A way to think of this is that all wrong values, less than zero, are treated as the same degree of wrongness. A graph of ReLU is depicted below:



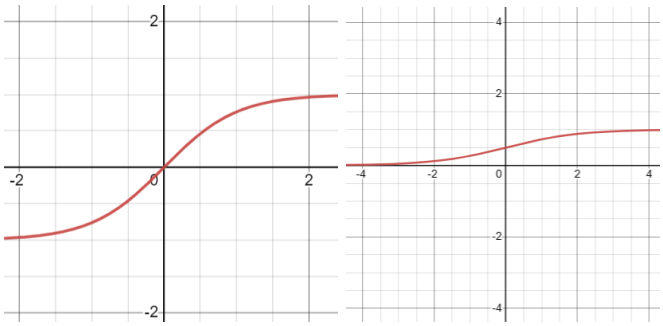
CNNs can be used on their own for HAR but they are often combined with an RNN. RNNs are useful because they can capture sequential relationships. An RNN node feeds its

output back into itself to be part of the next iteration's input. Traditional RNNs are the simplest form of this and only apply a weight to the recurrent input. More complicated RNNs such as GRUs and LSTMs implement gates to regulate the recurrent input in a more sophisticated fashion. The four common types of RNNs included in this report are traditional RNNs, GRUs, LSTMs, and BILSTMs. Gating the recurrent input addresses the vanishing/exploding gradient problem faced by traditional RNNs and allows the network to handle longer sequences. The vanishing/exploding gradient problem often occurs in a traditional RNN when it is trying to process an input with a long series. If the recurrent weight is greater than one, the recurrent value can grow much larger than the other input and dominate the problem. If the recurrent weight is less than one, it is diminished to insignificance by the end of the input series.

LSTMs maintain two recurrent values between steps: a short-term and long-term memory value. It uses a forget gate and input gate which modify the amount of recurrent memory values to forget. The input gate uses the modified long term memory, input, and short term memory to create a new long term memory. The result of the input gate is stored as the new long term memory and is also used to update the short term memory. GRUs, gated recurrent units, are a simplification of LSTMs which combines the input and forget gate and also only maintains one memory value between interactions. Bidirectional LSTMs, BILSTMs, ingest the sequence both forwards and backwards using a backward and forward layer. The diagram below depicts the architecture of a standard LSTM node.



LSTMs typically use the hyperbolic tangent activation function or tanh. Tanh takes in any input value and outputs a value between -1 and 1. It is similar in shape to the sigmoid activation function although the sigmoid activation function's output only ranges from 0 to 1. The equation for tanh is $y = (e^x - e^{-x}) / (e^x + e^{-x})$. In the case of LSTMs, the tanh function is used to represent the value of the proposed new long term and short term memory. These new values are combined with the sigmoid function to determine the new memories. Since the sigmoid function ranges from zero to one, it can be thought of as a percentage representing 'how much' of the tanh function should be saved. The equation for the sigmoid function is $y = e^x / (e^x + 1)$ The tanh and sigmoid functions are outlined below:



tanh (left) and sigmoid (right)

The final layer of all networks being tested is a fully connected layer using the softmax function. Softmax outputs a vector of values that range from zero to one and sum to one. These values represent the predicted probability of each output. It is called softmax because unlike the argmax function, it can provide 'softer' values between zero and one [Brownlee 2021]. This is valuable because it can be interpreted as the confidence of a classification whereas argmax outputs one classification and definitively claims it is the correct classification regardless of how close the other classifications were to being chosen. The equation for softmax can be expressed as: $y = e^x / sum(e^x)$.

The CNN-GRU model is optimized using the Keras Tuner tool in this paper. As briefly shown in [Joshi et. al. 2021], the Keras Tuner can be effectively used on CNN and LSTM networks. In [Joshi et. al. 2021] they used Keras Tuner to decide the number of nodes per layer, nodes for the input layer, number of convolution layers, and number of epochs. Keras Tuner provides three main algorithms for tuning: bayesian inference, random search, and hyperband.

III.2. Techniques that have been used by other researchers for the research problem

HAR is a long-standing classification problem and a lot of literature has been generated documenting different solutions. For this project, most techniques surveyed consist of deep learning; however, novel implementations using traditional statistical techniques are continuing to be researched. These approaches tend to be intended for situations where resources are restricted such as on smartphones or IOT devices in the case of [Lee et. al. 2020]. Listed below are several interesting techniques surveyed for this project.

III.2.1 AdaBoost Support Vector Machines

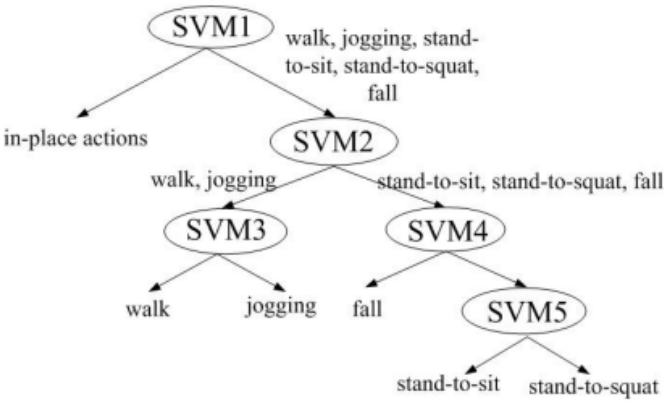
As discussed earlier, SVMs are binary classifiers: they find a boundary between two classes and maximize that boundary. For multiclass problems, multiple SVMs can be used. To go beyond linear problems and solve non-linear boundaries, kernels can be employed such as the Gaussian Radial Basis Function RBF listed below which creates a boundary around a central point.

$$K(x_1, x_2) = exp(-\frac{||x_1 - x_2||^2}{2\sigma^2})$$

Additionally, techniques can be applied to the data to transform it so that the data is easily divided along a hyperplane. [Yulita et. al. 2021] use SVMs for sensor data. Its novelty is the application of AdaBoost to the SVMs. AdaBoost is a technique that relies on training many weak learners. In AdaBoost, the order of SVM creation is important. If the currently generated SVM has errors in classification, those wrongly classified values are more heavily weighted when creating the next SVM. AdaBoost uses a voting system where SVMs that are best at classifying have more weight in the voting. They demonstrate good accuracy, 96.06 percent, although their dataset appears relatively simple.

III.2.2 Decision Trees

Another modern implementation of traditional machine learning is outlined in [Qian et. al. 2010]. In this paper, six activities are classified. This technique also uses SVMs but arranges them into a decision tree with the intention of recognizing multiple actions. A depiction of the decision tree from [Qian et. al. 2010] is outlined below.



Interestingly, this approach does not use sensor data but two-dimensional images. This paper does a lot of image preprocessing to extract a bounding box it describes as a ‘human blob’. The authors then analyze how the blob’s shape’s changes over time to make a classification. Although interesting, it highlights the amount of effort required to preprocess data that is unnecessary when using CNNs.

III.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks have a myriad of applications in deep learning. This is especially the case in the field of computer vision. This is largely due to the fact that they use filters instead of single tensors for node input unlike MultiLayer Perceptrons, MLPs. These filters allow nodes to capture patterns in the image. MLPs have largely been replaced by CNNs in HAR for this reason as stated in [Mutegeki and Han 2020]. [Wei et. al. 2021] research using only CNNs to identify human activities. In this paper, researchers use the popular AlexNet CNN architecture with a dataset of depth maps using Microsoft's Kinect camera. Interestingly, this approach does not rely on temporal relationships. Instead it only takes into consideration a single image when determining action. Importantly, the researchers also performed image prepping by subtracting the background input image. Additionally, the depth map image provides more easily understood input than a traditional image. In a traditional image, the depth of each pixel along the z axis is unknown.

III.2.4 Long Short-Term Memory Networks (LSTMs)

Long short-term memory networks, LSTMs, take an ordered sequence as input. They are considered a recurrent network which means that the output of a node is fed back into its input for the next state. LSTMs extend RNNs one step further with a long term and short term memory stored between steps. These two memory units also contribute to the nodes input and are updated each step. This network is considered gated which means that the input and output of the memory units are regulated using the sigmoid and tanh functions to determine how much the node should rely on 'remembering' previous steps and how quickly it should 'forget' them. [Moradi et. al. 2022, Perez-Gamboa et. al. 2021] both include standalone LSTM networks in their comparisons for learning. In both cases LSTM performs well but not as well as LSTM combined with CNN. Variations of the LSTM include the GRU and BILSTM which are explained in more detail in the preceding sections.

III.2.5 CNN and LSTM Hybrid (CNN-LSTM)

Combining a CNN with an LSTM is popular in academic research and is the focus of this paper. This approach aims to combine the strengths of CNNs with the strengths of LSTMs. All models surveyed start with the CNN being used to extract features. Its output is then fed to the LSTM network which is effective at understanding the temporal relationship between inputs. This output is then fed to a softmax layer that makes the classification. This approach has a lot of interesting promise but the size and architecture of the networks presented in papers are not well explained or justified.

III.3. Relevant technologies that would be useful to this research

The writing, testing, and analysis for this report is performed with Python. Keras, Tensorflow, Numpy, Scikit-learn, Matplotlib, and Pandas are used for modifying and formatting the data. Additionally, the Keras Tuner tool is used for optimizing hyperparameters. Two datasets are used for testing: WISDM and MHEALTH.

III.4 Algorithms, Process modules, and solution methods

Adam (Adaptive Moment Estimation), is used as the optimization function for all training in this paper. It is a common optimization function and an extension of traditional stochastic gradient descent. Adam differs from stochastic gradient descent in that it does not just use one learning rate but uses a different learning rate for each parameter that can change as the training progresses.

LSTMs use an algorithm for regulating recurrent values and maintaining two memories: one short-term and one long-term memory. The algorithm first multiples the long-term memory by the forget gate's weight. The forget gate's weight is determined by summing the weighted input and weighted short-term memory and then applying those results to the sigmoid function. A value of 1 corresponds to being perfectly remembered and a value of 0 means the long-term memory is completely forgotten.

After being altered by the forget gate, the long term memory is combined with the result of the input gate to form the new long-term memory. The input gate uses the short-term memory and input to determine a new value (using tanh) and the amount that new value should be remembered (using sigmoid). The algorithm then uses the long term memory, short-term memory and input to generate the short term memory. If this is the last step in the input sequence, the short term memory is the output. A diagram of an LSTM node is included in section III.1 for further clarification. GRUs and BILSTMs use variations of this algorithm with GRUs simplified to not maintain a long-term memory and BILSTMs modified to handle two directions of input.

The first metric used for this paper is accuracy. Accuracy is the most commonly used metric in surveyed literature and is often the main measure of a model's performance. Accuracy is defined as: $\frac{Number\ of\ Correct\ Predictions}{Total\ Predictions}$ and is calculated based on all of the models predictions.

Precision is defined as $\frac{True\ Positives}{True\ Positives + False\ Positives}$. In other words, it is a measure of how many samples predicted positive are truly positive. Precision is a useful metric when false positives are very costly. A real world example could be identifying safe-to-eat mushrooms. It is very important to not accidentally identify a mushroom as safe to eat when it is not.

Recall is defined as $\frac{True\ Positives}{True\ Positives + False\ Negatives}$. This measures how many of the positive samples were wrongly identified as negative. This measure is useful when a false negative is very costly. An example of this could be screening people for a serious disease such as Ebola. It is incredibly important that no-one with Ebola is wrongly identified as negative.

The training time measure is trivial and does not require algorithms beyond averaging the training time of each epoch. The averaging is probably unnecessary as most epochs are the same or one milliseconds slow. This measure is not precise and arbitrarily tied to my machine's resources. The values can range slightly based on other processes running on my machine and contending for resources. The importance of this measure is not the absolute time of each architecture's training but their value relative to each other.

The algorithm used for hyperparameter tuning is called hyperband. Hyperband tuning uses a 'sports championship style bracket' for comparing different combinations of hyperparameters according to the official Tensorflow documentation. It trains each combination of parameters over a few epochs and then the best performing models advance to the next round of training and comparison. There are various customization options for training such as setting thresholds for early quitting and limiting the number of epochs per round. One advantage mentioned in the Keras documentation is that Hyperband tuning is memory efficient as it only trains for a few epochs each round and quickly widdles down the number of models in contention.

The goal of this paper is to compare how different layer types affect performance; therefore, simplicity in network complexity was prioritized. The networks without a CNN layer prepended consist of one layer for their tested node type with 25 units, a dropout layer of 20 percent, a flatten layer, and a softmax layer for final classification. CNN prepended networks consist of a 25 unit CNN layer, a dropout layer of 20 percent, 25 unit layer of the test node type, a dropout layer of 20 percent, a flatten layer, and a final softmax classification layer. Examples of a CNN and CNN-GRU network are depicted below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              
=================================================================
 conv1d (Conv1D)             (None, 98, 25)            

 dropout (Dropout)           (None, 98, 25)            

 flatten (Flatten)           (None, 2450)              

 dense (Dense)               (None, 6)                 

=================================================================
Total params: 14,956
Trainable params: 14,956
Non-trainable params: 0
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              
=================================================================
 conv1d (Conv1D)             (None, 98, 25)            

 dropout (Dropout)           (None, 98, 25)            

 gru (GRU)                   (None, 25)                

 dropout_1 (Dropout)         (None, 25)                

 flatten (Flatten)           (None, 25)                

 dense (Dense)               (None, 6)                 

=================================================================
Total params: 4,306
Trainable params: 4,306
Non-trainable params: 0
```

IV. Approach(s)

IV.1. Methodologies I applied in this research

This project implements deep learning networks using Keras. My implementation consists of following network architectures:

- Traditional RNN
- CNN
- GRU
- LSTM
- BILSTM
- CNN-CNN
- CNN-RNN
- CNN-GRU
- CNN-LSTM
- CNN-BILSTM

A survey of literature on deep learning in human activity recognition inspired an interest in performing a comprehensive benchmarking of performance. Surveyed literature proposes novel models that combine CNNs and different types of RNNs but do not provide a comprehensive comparison. It is frustrating to read that a model performs better than one or two selected architectures to compare against without having the entire context. Additionally, the surveyed literature often limits the metrics of performance and only focuses on accuracy. This paper addresses that lack of comparison, and provides a benchmarking of the commonly used deep learning architectures discussed in the literature. This paper provides more subtlety to the performance comparison and shows that selecting the 'best' architecture is dependent on what metrics are most valued. This paper measures the performance of these models using accuracy, precision, recall, and training time. We then apply hyperparameter tuning to the CNN-GRU model to briefly explore how tuning improves performance.

IV.2. Techniques I used to solve the problem

To provide a benchmarking comparison of the ten architectures, many training and testing iterations were performed. To benchmark fairly, it is important to minimize the number of differences between networks and use default settings whenever possible to capture how they most commonly perform. One difficult difference was deciding the kernel size of CNN nodes. A CNN's kernel size can greatly impact performance whereas a GRU has no concept of a kernel size. I tested CNN kernel sizes from one to five using the WISDM dataset and selected the most successful size by accuracy which was three. After selecting three, I kept this kernel size consistent throughout all following tests. Other parameters were kept consistent between tests. For example, all networks have 25 nodes per layer, use a dropout layer of twenty percent, and use a softmax layer for final classification.

Data preparation also required searching for techniques that performed well. I initially intended to only use the WISDM dataset with no noise. Working with the dataset, I found that it was not particularly interesting and opened up additional questions I had not thought of during my proposal. For example, how do the networks perform when the input is noisy? The WISDM

dataset only has one accelerometer measuring x, y, and z movement. How do the networks perform with different types of sensors? WISDM only includes six activities. How do the networks change in performance as the number of activities increase? The WISDM sampling frequency is 20Hz, what happens when the sampling frequency is changed?

To answer these questions I sought out the MHEALTH dataset. This dataset consists of three accelerometers, two gyroscopes, an electrocardiogram, and two magnetometers. Additionally MHEALTH has twelve activity classifications and samples at 50Hz. Testing the networks against both datasets provided more robust benchmarking. For example, similar results found using both datasets suggest they are not simply dependent on one specific set of data. To test the impact of six classifications versus twelve, I ran a test where only one arm accelerometer was included in the MHEALTH dataset to mimic the one arm accelerometer available in the WISDM dataset.

The technique I found for preparing noisy data is from [Ackermann 2021]. This technique removes records with missing data and then slices the dataset into timeframe chunks. The truth label is set to the most common label in the time slice. This results in many samples that consist of one activity but also includes inputs where the beginning of the input is the previous activity or the activity ends and the last segment of input is the starting of the next activity. This makes the data resemble the noise of real life. Additionally the MHEALTH dataset has an oversized amount of unlabelled samples where participants were not performing any action. This unbalanced amount of unlabelled sampling was removed from the dataset for testing.

The techniques for analyzing and comparing the network performances are straightforward. After 1000 epochs of training, the models still had some variation in their performance between epochs. To minimize the impact of variation, comparisons were made using an average of the last 50 epochs. Additionally a dropout of 20 percent was applied after neural layers for training. This was not strictly necessary for comparison but helps provide better results. This is a common technique to make a layer not over rely on one subset of its nodes and perform better against validation data. This slightly skews performance metrics against the training data but is acceptable because we are more concerned with how the networks perform against the validation data.

V.3. Processes I engaged in this research

All code was written using Python and various libraries including Keras, Tensorflow, Numpy, Scikit-learn, Matplotlib, and Pandas. To run the tests, I split the data sets into 80 percent training and 20 percent validation for 1000 epochs. The selection of 1000 epochs was arbitrary and probably excessive for training length. The training history for each test run was saved to a CSV file and the models' weights were saved to various .pb files according to the Keras API. The weights allowed loading of the model after training to retest performance. The CSV training histories were used with Numpy, Pandas, Matplotlib, and Scikit-learn to read the CSV into a dataframe, generate graphs, and measure metrics. Keras Tuner wasused for the hyperparameter tuning. The selected hyperparameters to test are arbitrarily based on the performance of the CNN-GRU network in previous experiments. Tables shown in this report are compiled using Google Sheets.

The primary constraint during testing was to ensure my results were reasonably in-line with surveyed results. For example, it was expected that prepending the CNN layers to RNN layers would improve performance but it was unknown to what degree there would be improvement. The testing was judged on whether it provided a more subtle understanding of how the networks perform. This paper found several nuances such as that additional sensors provide benefit when CNNs are included, CNN based networks often fit the training data well but degrade against validation data faster than RNN based networks, and more complicated RNNs perform best when the input size is large.

IV.4. Facilities and supplies used for this research

No special facilities or supplies were used. All tests were run on my personal computer. To speed up testing, Keras was run using the Cuda drivers to offload training to a NVIDIA GeForce GTX 1070 GPU.

V. Work Conducted

V.1. Task performed in this research

V.1.1 Prepare and Understand the WISDM Dataset

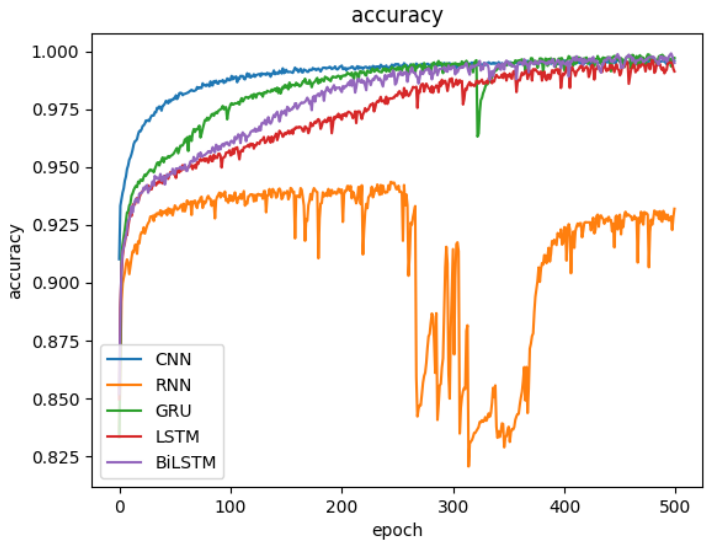
The first step for testing involved understanding and preparing the WISDM dataset. I initially intended to only test without noise. No noise required simply iterating over the dataset and collecting samples where a contiguous set of time steps all shared the same label and was as long as my desired size.

V.1.2 Implement tests for WISDM Dataset

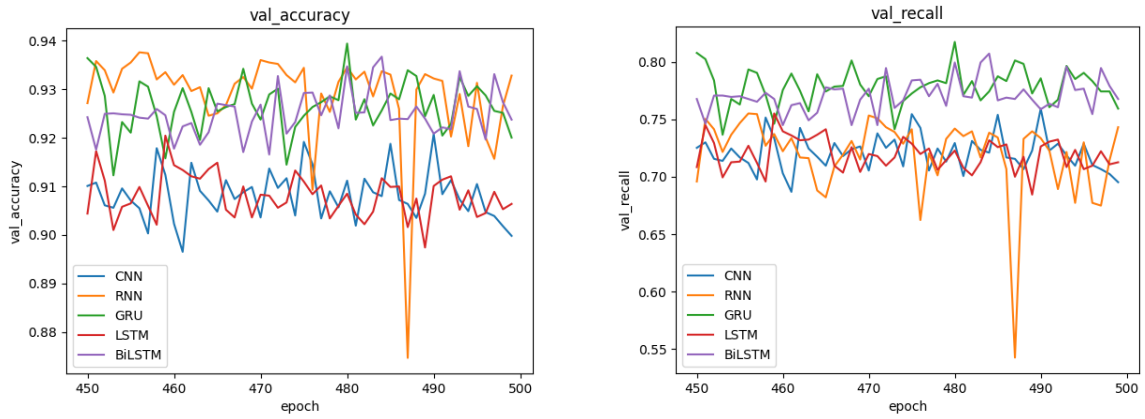
This step involved testing the WISDM dataset against the CNN, RNN, GRU, LSTM, and BILSTM architectures. My initial test involved no noise and time frames of 100 time steps. This testing showed results that are similar to those shown in the results section below although they are not outlined in the report as they were not particularly interesting. 100 time steps was an arbitrary length that corresponds to a real-time length of five seconds. This step required setting up the code to generate the targeted model, run the test, save model weights, save training history, and generate a confusion matrix. This step provided some insight into the performance of the different architectures. I was surprised by how close the networks performed when measuring accuracy

V.1.3 Graph Results of Initial Tests:

This step consists of writing functions that graph results using Numpy, Pandas, Matplotlib, and Scikit-learn. The code for testing the networks automatically saves the training results and learned weights. This allows graphing functions to be generic. The graphing functions load the specified training file, convert it into a dataframe, and then graph the corresponding data frame components. Depicted below are several interesting graphs and a confusion matrix from the initial tests.



The accuracy of the traditional RNN noticeably drops during training and recovers between epochs 250 and 400. This is likely because of two factors: a 20 percent dropout layer applied during training and the vanishing/exploding gradient problem.



These two graphs show performance of the networks against validation data. Interestingly, the traditional RNN, GRU, and BILSTM all perform similarly well averaging approximately 93 percent accuracy. When measuring recall, the GRU and BILSTM perform best with performance ranging between 80 and 75 percent while the traditional RNN performs worse with a success range between 70 and 75 percent. This illustrates a common problem with literature surveyed where only accuracy is considered for network performance.

V.1.4 Implement CNN-Prepended Networks using WISDM Dataset and Graph Results

This step involved prepending each network with a CNN layer of 25 nodes. Each CNN node has a filter size of three. I based the filter size on testing different kernel sizes between one and five and selecting the best performing size based on accuracy. A dropout layer of 20 percent was included after the CNN layer. Interestingly, these networks perform better than the simple networks but only by a few percent. This suggests that in compute constrained machines, the slightly less accurate models may be more favorable than a deeper network.

V.1.5 Include MHEALTH dataset and noisy input

After testing with WISDM, I had more questions I had not considered during my project proposal. I was unsure whether the similar performances could be attributed to only having six classifications. I was also curious if there would be more interesting results with a different type of sensor or more sensors than just one accelerometer. Searching online I found the MHEALTH dataset. This dataset includes several accelerometers, electrocardiograms, gyroscopes, and magnetometers. Additionally, it includes twelve activities.

During this time I also considered that noisy input could have an impact on performance. A technique I found was in [Ackermann 2021]. As discussed in previous sections, this approach removes records with missing data and then slices the dataset into timeframe chunks. This approach allows robust data input and resembles the noise of real life. Many inputs include just one activity but some are also created where a previous activity is finished before starting the primary activity or the primary activity is completed and another activity is started.

V.1.6 Test non-noisy input and simplified MHEALTH

To test how the number of classifications impact performance I ran the MHEALTH dataset with no noise. MHEALTH includes 12 classifications whereas WISDM only includes 6. I also limited the input to only one arm accelerometer to match the one accelerometer in the WISDM dataset.

V.1.7 Test noisy input for MHEALTH with only one accelerometer and with all sensors

This step required running tests for all architectures using noisy input. To keep the tests between datasets similar, I restricted the MHEALTH dataset to only using one arm accelerometer similar to WISDM. The final test I ran used all sensors and noise. This test yielded interesting results showing the benefit of prepending a CNN layer as outlined in the results section.

V.1.8 Hyperparameter Tuning

This final test used the CNN-GRU network for hyperparameter testing. The CNN-GRU network was selected as it performed well throughout the tests. Additionally, it trains faster than networks that use LSTM and BILSTM. This step required reading the Keras Tuner documentation and official tutorial to understand how it functions. Keras Tuner offers three techniques for tuning: bayesian inference, random search, and hyperband. Hyperband was selected for testing as it was the most commonly cited approach. I also spent some time understanding how the parameters such as max epochs and epoch iterations impact the tuning. I decided on testing the CNN layer size, CNN kernel size, and GRU layer size. I also reduced the input size down to 25 steps (0.5 seconds) in order to compare against a baseline CNN-GRU that performed worse than when the input was 100 time steps. 100 time steps could have been used but reducing it to 25 increased the range of possible improvement. The baseline CNN-GRU only had CITE performance on validation precision; therefore, I selected precision as the metric to tune for. After 234 trials the tuning ended with the best precision racing 93.2 percent. The details of this test are outlined in the results and analysis section.

V.1.5 Graphing and collating results for report

This final step was rather trivial and involved using Scikit-learn, Numpy, Matplotlib, and Pandas to extract data from the training histories to collate data for the report. Much of this code was written earlier in the semester. For much of the performance measurement, I included an average of the last 50 epoch values in comparison. Results were collated in Google Sheets for the report.

V.2 Schedule, timeline, and milestones

Order	Dates	Task/Activity	Prerequisites	Results Obtained
1	01/23 -2/13	Research potential topics, select and research CNN-RNN benchmarking, and write project proposal.	None	Project selected and outline of planned work created
2	2/13 - 2/20	Prepare and understand WISDM dataset.	Topic selected	WISDM dataset ready to be used as input for tests
3	2/20 - 3/6	Implement non-noisy tests for the WISDM dataset. This includes writing code for creating the model, preparing data, training, saving results to static files, and generating a confusion matrix for the final weights against validation data.	WISDM dataset ready for use	Initial baseline tests conducted for different architectures
4	3/6 - 3/13	Write code for graphing results, get average performance, and analyze results.	Results of testing obtained and graphed	Able to display average performance, training over time, performance based on an arbitrary metric, etc.
5	3/13 - 3/22	Implement CNN prepended networks and test performance.	Simple versions of networks available to be extended	CNN prepended network performance measured and available to compare against simple networks

6	3/22 - 4/3	Prepare and understand MHEALTH dataset. Setup data preparation for noisy input.	None, I decided to add MHEALTH to make benchmarking more robust.	Experiments prepared that make the benchmarking more robust i.e. noisy data, more sensors, more classification categories.
7	4/3 - 4/10	Test non-noisy input for simplified MHEALTH dataset and compare against WISDM.	MHEALTH dataset prepared	Comparison of how networks perform between the two datasets
8	4/10-4/12	Test using all sensors with MHEALTH dataset.	MHEALTH dataset prepared	Comparison of how adding more sensors impacts network performance available for analysis
9	4/12 - 4/24	Learn how to use Keras Tuner, select parameters, and write code for hyperparameter tuning. Tune CNN-GRU model.	Tests ran and selected CNN-GRU for good tuning candidate	Impact of tuning measured on CNN-GRU network
10	4/24-4/27	Graph and collate results for report	All testing complete	Graphs and tables available for adding to report
11	4/27 - 5/3	Write final report and presentation	All testing and graphs created	Project complete

VI. Results and Analysis

VI.1 Results

VI.1.1 Methods and Techniques Used

This project utilizes Python scripts to execute all steps of the experiments and analysis including data loading, preparation, model testing, and performance analysis. The Keras library, which is built on top of TensorFlow, is utilized for creating the models and training them. Additional libraries such as Pandas, Numpy, Matplotlib and Scikit-learn are used to prepare and analyze the results.

Tests involve measuring the performance of each network architecture against the same test and validation data. Their results are then compared to see how well they perform across different metrics. All convolutional and recurrent layers include a twenty percent dropout rate during training. This is a common technique to make a layer not over rely on one subset of its nodes and perform better against validation data. It should be noted that this also slightly skews performance metrics against the training data. Testing against the validation data does not include the drop-out layer; therefore, does not skew validation results.

Two techniques are used for data preparation. The first is for non-noisy data and is trivial. It involves iterating over the data and selecting contiguous sections that all have the same label and are as long as our selected input length. The next type of data preparation is from [Ackermann 2021]. This technique removes records with missing data and then slices the dataset into timeframe chunks. This approach allows robust data input and resembles the noise of real life. Many frames include just one activity but some frames are also created where a previous activity is finished before starting the primary activity or the primary activity is completed and another activity is starting.

Although the model's are asymptotically reaching their performance limit after 1000 epochs, as shown in the Appendix's graphs, there is still some variation between the later epochs. To minimize this variance, we use the average performance of the last 50 epoch weights to minimize this variance.

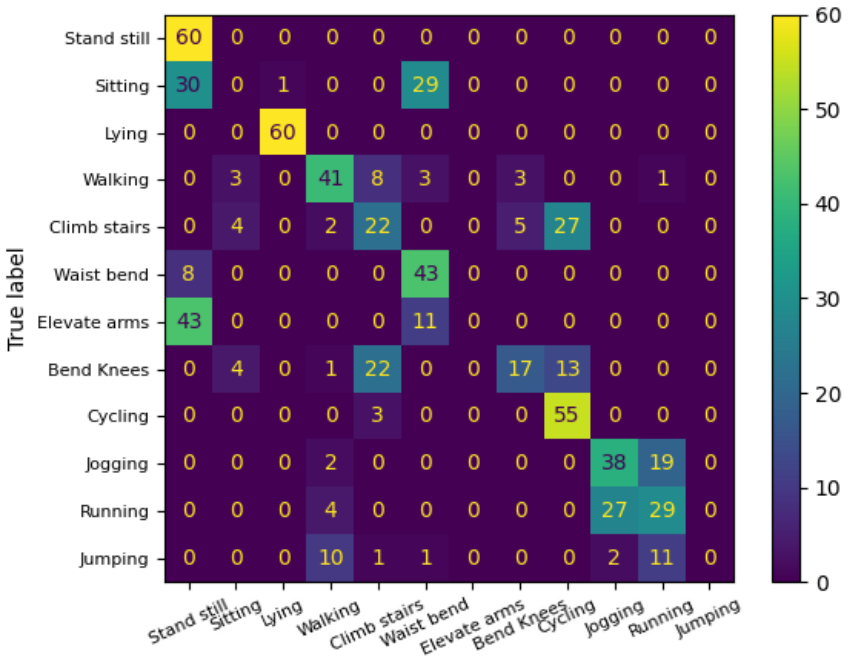
VI.1.1 Results MHEALTH Dataset Using One Accelerometer and No Noise

The first set of benchmarking tests use the MHEALTH dataset. This dataset contains twelve activity classifications and has a sampling frequency of 50Hz. Time frames of 100 samples (two seconds) are the input. Twenty percent of the samples are reserved for validation testing. The samples do not include noise. In other words, every step in the sample's 100 steps has the same activity label. Additionally, only the left arm accelerometer measuring x, y, and z movement are used. This is done to make the samples more similar to the WISDM dataset which only includes one accelerometer. One thousand epochs of training are applied to each network. The table below summarizes the averages for the last 50 epochs of training.

Average of Last 50 Epochs for MHEALTH Dataset (100 Time Steps / 1 Accelerometer / No Noise)								
Model	**Accuracy**	**Precision**	**Recall**	**Loss**	**Val Accuracy**	**Val Precision**	**Val Recall**	**Val Loss**
CNN	0.999	0.996	0.996	0.017	0.959	0.735	0.730	2.285
RNN	0.934	0.682	0.255	1.220	0.936	0.725	0.268	1.178
GRU	0.994	0.965	0.961	0.101	0.968	0.797	0.787	0.960
LSTM	0.996	0.976	0.974	0.069	0.959	0.737	0.733	2.329
BILSTM	0.998	0.989	0.988	0.033	0.967	0.785	0.784	1.332
CNN-CNN	0.999	0.994	0.994	0.019	0.966	0.779	0.777	3.907
CNN-RNN	0.978	0.866	0.843	0.393	0.970	0.816	0.793	0.496
CNN-GRU	0.997	0.979	0.978	0.061	0.977	0.852	0.850	0.524
CNN-LSTM	0.998	0.984	0.984	0.042	0.973	0.823	0.820	0.753
CNN-BILSTM	0.998	0.990	0.990	0.026	0.969	0.798	0.797	1.429

Unsurprisingly, all of the networks performed well in accuracy, precision, and recall

against the training data with the exception of the RNN network. The RNN network performs significantly worse than the other networks when measuring recall. Recall is defined as $recall = \frac{true\ positive}{true\ positive + false\ negatives}$. The confusion matrix for the RNN is outlined below.



It is notable that sitting, elevated arms, and jumping appear completely unlearned as the network incorrectly identifies all of these incorrectly. This high number of false negatives appears to be driving down the recall of the network.

The performance of networks against the validation data provides the most interesting comparison. The CNN and CNN-CNN networks are highly performant against the training data but depreciate in performance more than the other networks against the validation data. Unsurprisingly, the CNN-prepended networks consistently perform better than the un-prepended networks. This shows that the CNN prepend to a RNN network does indeed provide better learning as discussed in the following analysis section. The accuracy amongst all models was very tight with a range of 4.1 percent. Precision showed a greater range of 12.7 percent. Recall showed a similar range of 12.1 excluding the RNN network which has an outlier value of 26.8 percent. The validation results are depicted below and ranked by performance:

Val Accuracy	
CNN-GRU	0.977
CNN-LSTM	0.973
CNN-RNN	0.970
CNN-BILSTM	0.969
GRU	0.968
BILSTM	0.967
CNN-CNN	0.966
LSTM	0.959
CNN	0.959
RNN	0.936

Val Precision	
CNN-GRU	0.852
CNN-LSTM	0.823
CNN-RNN	0.816
CNN-BILSTM	0.798
GRU	0.797
BILSTM	0.785
CNN-CNN	0.779
RNN	0.770
LSTM	0.737
CNN	0.725

Val Recall	
CNN-GRU	0.850
CNN-LSTM	0.820
CNN-BILSTM	0.797
GRU	0.787
BILSTM	0.784
CNN-RNN	0.793
CNN-CNN	0.777
LSTM	0.733
CNN	0.730
RNN	0.268

VI.1.2 MHEALTH Dataset Using One Gyroscope and Noise

The next test uses the same parameters but allows noise to be included in the samples. The results of this testing and the ranked validation data performance are outlined below:

Average of Last 50 Epochs for MHEALTH Dataset (100 Time Steps / 1 Accelerometer / Noise)								
Model	Accuracy	Precision	Recall	Loss	Val Accuracy	Val Precision	Val Recall	Val Loss
CNN	0.999	0.995	0.994	0.024	0.960	0.743	0.737	2.568
RNN	0.928	0.762	0.099	1.741	0.929	0.838	0.110	1.583
GRU	0.992	0.951	0.946	0.136	0.972	0.819	0.810	0.812
LSTM	0.996	0.977	0.975	0.064	0.963	0.760	0.754	2.000
BILSTM	0.996	0.977	0.975	0.066	0.970	0.805	0.802	1.285
CNN-CNN	0.999	0.992	0.992	0.023	0.968	0.795	0.790	4.557
CNN-RNN	0.952	0.761	0.541	0.857	0.955	0.798	0.556	0.856
CNN-GRU	0.997	0.981	0.980	0.053	0.971	0.815	0.812	0.891
CNN-LSTM	0.997	0.982	0.981	0.048	0.971	0.819	0.809	1.066
CNN-BILSTM	0.999	0.991	0.991	0.025	0.968	0.795	0.792	1.479

Val Accuracy	
GRU	0.972
CNN-GRU	0.971
CNN-LSTM	0.971
BILSTM	0.970
CNN-BILSTM	0.968
CNN-CNN	0.968
LSTM	0.963
CNN	0.960
CNN-RNN	0.955
RNN	0.929

Val Precision	
RNN	0.838
CNN-LSTM	0.819
GRU	0.819
CNN-GRU	0.815
BILSTM	0.805
CNN-RNN	0.798
CNN-CNN	0.795
CNN-BILSTM	0.795
LSTM	0.760
CNN	0.743

Val Recall	
CNN-GRU	0.812
GRU	0.810
CNN-LSTM	0.809
BILSTM	0.802
CNN-BILSTM	0.792
CNN-CNN	0.790
LSTM	0.754
CNN	0.737
CNN-RNN	0.556
RNN	0.110

The most interesting result of this test is that the results are only slightly different from the testing with no noise. For example the GRU, LSTM, and CNN-CNN networks all performed approximately two percent better on the noisy data when measuring validation precision. The performance difference for validation accuracy between noisy and non-noisy data was less than one percent for all models except the RNN and CNN-RNN models. The further degradation of these two models' performance makes intuitive sense as we are now introducing noise.

Traditional RNN nodes cannot regulate their recurrent inputs therefore they cannot adapt well to changes in the activity.

The results of the other models suggest that they handle noisy activities well. This is inline with [Moradi et. al. 2022] and [Mutegeki and Han 2020] which state deep learning techniques perform robustly against input that contains noise and does not match the training data perfectly.

VI.1.3 WISDM Dataset Using One Gyroscope and Noise

This test uses the WISDM dataset which consists of six activities, one accelerometer, and noisy input. Similar to the previous test, this approach uses the technique in [Ackermann 2021] to allow noise in the data. One notable difference is that WISDM uses a sampling frequency of 20Hz. As a result, 100 time steps corresponds to five seconds. The results of this testing are outlined below:

Average of Last 50 Epochs for WISDM Dataset (100 Time Steps/ 1 Accelerometer / Noise)								
Model	Accuracy	Precision	Recall	Loss	Val Accuracy	Val Precision	Val Recall	Val Loss
CNN	0.993	0.981	0.978	0.063	0.905	0.717	0.708	2.185
RNN	0.842	0.629	0.122	1.300	0.838	0.753	0.054	1.314
GRU	0.998	0.994	0.993	0.024	0.918	0.755	0.751	1.976
LSTM	0.997	0.993	0.991	0.028	0.920	0.761	0.756	1.868
BILSTM	0.999	0.996	0.996	0.014	0.913	0.740	0.737	2.393
CNN-CNN	0.998	0.994	0.994	0.018	0.925	0.776	0.771	1.889
CNN-RNN	0.908	0.787	0.613	0.864	0.902	0.760	0.600	0.933
CNN-GRU	0.998	0.994	0.993	0.020	0.940	0.822	0.816	1.122
CNN-LSTM	0.998	0.995	0.994	0.019	0.939	0.817	0.814	1.348
CNN-BILSTM	0.999	0.998	0.997	0.008	0.933	0.800	0.796	1.460

Val Accuracy	
CNN-GRU	0.940
CNN-LSTM	0.939
CNN-BILSTM	0.933
CNN-CNN	0.925
LSTM	0.920
GRU	0.918
BILSTM	0.913
CNN	0.905
CNN-RNN	0.902
RNN	0.838

Val Precision	
CNN-GRU	0.822
CNN-LSTM	0.817
CNN-BILSTM	0.800
CNN-CNN	0.776
LSTM	0.761
CNN-RNN	0.760
GRU	0.755
RNN	0.753
BILSTM	0.740
CNN	0.717

Val Recall	
CNN-GRU	0.816
CNN-LSTM	0.814
CNN-BILSTM	0.796
CNN-CNN	0.771
LSTM	0.756
GRU	0.751
BILSTM	0.737
CNN	0.708
CNN-RNN	0.600
RNN	0.054

The results for this test are similar to those from the MHEALTH dataset. Once again, the CNN and CNN-CNN networks fit the training data very well but depreciate more than the other networks against validation data. The CNN-GRU, CNN-LSTM, and CNN-BILISTM networks all performed the best in all validation metrics.

VI.1.4 MHEALTH Dataset Using All Sensors and Noise

The next benchmark test uses the MHEALTH dataset but expands the input to include all available sensors. Whereas previous tests were limited to one accelerometer, this test includes the following:

- Chest, left ankle, and right lower arm accelerometers
- Electrocardiogram
- Left ankle and right lower arm gyroscopes
- Left ankle and right lower arm magnetometer

This test included noise and 100 time steps per input. The results are outlined below:

Average of Last 50 Epochs for WISDM Dataset (100 Time Steps/ All Devices / Noise)								
Model	Accuracy	Precision	Recall	Loss	Val Accuracy	Val Precision	Val Recall	Val Loss
CNN	1.0000	0.9999	0.9999	0.0002	0.9905	0.9386	0.9385	0.4052
RNN	0.9617	0.8385	0.6204	0.7646	0.9415	0.6607	0.4925	1.4295
GRU	0.9995	0.9969	0.9966	0.0158	0.9702	0.8067	0.8056	1.2230
LSTM	0.9995	0.9972	0.9966	0.0143	0.9678	0.7938	0.7846	1.3606
BILSTM	0.9999	0.9992	0.9991	0.0034	0.9882	0.9244	0.9225	0.4497
CNN-CNN	0.9999	0.9996	0.9996	0.0024	0.9936	0.9582	0.9581	0.3034
CNN-RNN	0.9754	0.8767	0.7914	0.4345	0.9672	0.8040	0.7591	0.6176
CNN-GRU	0.9996	0.9978	0.9976	0.0091	0.9912	0.9435	0.9427	0.3159
CNN-LSTM	0.9998	0.9984	0.9983	0.0070	0.9946	0.9648	0.9645	0.1835
CNN-BILSTM	0.9999	0.9995	0.9995	0.0016	0.9949	0.9670	0.9666	0.1762

Val Accuracy	
CNN-BILSTM	0.9949
CNN-LSTM	0.9946
CNN-CNN	0.9936
CNN-GRU	0.9912
CNN	0.9905
BILSTM	0.9882
GRU	0.9702
LSTM	0.9678
CNN-RNN	0.9672
RNN	0.9415

Val Precision	
CNN-BILSTM	0.9670
CNN-LSTM	0.9648
CNN-CNN	0.9582
CNN-GRU	0.9435
CNN	0.9386
BILSTM	0.9244
GRU	0.8067
CNN-RNN	0.8040
LSTM	0.7938
RNN	0.6607

Val Recall	
CNN-BILSTM	0.9666
CNN-LSTM	0.9645
CNN-CNN	0.9581
CNN-GRU	0.9427
CNN	0.9385
BILSTM	0.9225
GRU	0.8056
LSTM	0.7846
CNN-RNN	0.7591
RNN	0.4925

This test showed significant improvement for the CNN prepended networks in comparison to the previous tests only using the accelerometer. Their accuracy against validation data improved between two and three percent in each case (excluding CNN-RNN). This may not sound significant but their accuracies were all already above 96 percent. The difference in performance is much more pronounced in precision and recall. For example, the CNN-BILSTM network improved 17.2 and 17.4 percent in precision and recall when including all sensors. These improvements are outlined in the tables below:

Comparison of MHEALTH Acceleometer v. All Sensors (100 time steps / noise)

	Validation Accuracy			Validation Precision			Validation Recall		
Model	All Sensors	Accel.	Difference	All Sensors	Accel.	Difference	All Sensors	Accel.	Difference
CNN-BILSTM	0.995	0.968	0.027	0.967	0.795	0.172	0.967	0.792	0.174
CNN-LSTM	0.995	0.971	0.023	0.965	0.819	0.146	0.964	0.809	0.156
CNN-CNN	0.994	0.968	0.025	0.958	0.795	0.163	0.958	0.790	0.168
CNN-GRU	0.991	0.971	0.020	0.943	0.815	0.128	0.943	0.812	0.131
CNN	0.991	0.960	0.030	0.939	0.760	0.178	0.939	0.737	0.202
BILSTM	0.988	0.970	0.018	0.924	0.805	0.119	0.923	0.802	0.121
GRU	0.970	0.972	-0.001	0.807	0.819	-0.012	0.806	0.810	-0.005
LSTM	0.968	0.963	0.005	0.804	0.798	0.006	0.785	0.754	0.030
CNN-RNN	0.967	0.955	0.013	0.794	0.743	0.051	0.759	0.556	0.203
RNN	0.941	0.929	0.012	0.661	0.838	-0.178	0.493	0.110	0.383

The noticeable improvements appear dependent on CNN layers being present in the networks. Non-CNN networks did not have notable improvement and in some cases decreased in performance. The implication is that more data is as valuable as the network's ability to parse meaningful patterns from it. RNN networks perform poorly at finding non-temporal patterns. In contrast, CNNs are effective at extracting the patterns. The CNN prepended networks have the best performance because they combine the benefits of extracting both spatial and temporal values.

VI.1.5 Hyperparameter Testing

As part of this paper's thesis to benchmark different architectures, it is worthwhile to briefly explore the impact of tuning a selected architecture and demonstrate potential improvements. Hyperparameters refer to the settings of the network itself. This can include settings such as the number of layers, layer activation function, layer size, optimization function, and kernel size. Keras Tuner provides three techniques for searching for ideal hyperparameters: bayesian inference, random search, and hyperband. For this paper we use hyperband tuning. Hyperband tuning uses a 'sports championship style bracket' for comparing different combinations of hyperparameters according to the official Tensorflow documentation. It trains each combination of parameters over a few epochs and then the best performing models advance to the next round of training and comparison.

After reviewing the networks and their performance in the previous tests I selected the CNN-GRU network for hyperparameter testing. This network consistently performed well in the tests and trains faster than LSTM and BILSTM networks. Three hyperparameters were identified for tuning. The first is the number of units in the CNN layers. I tested changing this value from 25 to 50 using a step of five. I also tested the kernel size for the CNN nodes with values ranging from one to five with a step of one. The last tuning parameter is the number of GRU units with a range of 25 to 50 and a step of five. The documentation suggests running the entire hyperband search as many times as resources allow; therefore, this paper ran the hyperparameter searching five times. To reduce the CNN-GRU's baseline performance, I reduced the time steps per input to 25 (0.5 seconds in real time). 100 time steps could have been used but, reducing it to 25 increased the range of possible improvement. The baseline CNN-GRU only had 79.4 percent validation precision; therefore, I selected precision as the targeted metric.

After 243 trials, the network converged on an architecture that was achieving 93.2 percent validation precision. Keras Tuner found this precision using a network of 50 CNN nodes, a kernel size of 4, and 50 GRU units. The increasing of the layer sizes to their maximum value is not surprising as making the network larger can often make the results better. It is surprising that a kernel size of four was selected over a kernel size of three. Kernel sizes of three were used in all of this paper's benchmarking tests. This size was selected based on running the CNN network using kernel sizes from one to five and selecting the most performant in terms of validation accuracy. For hyperparameters where a good value is not readily apparent (such as kernel size) tuning can be very beneficial. Without the constraint of very long tuning times, additional experimentation with tuning would have been beneficial. It should be noted that if another metric other than validation precision was targeted for optimization, Keras Tuner might offer a slightly different network architecture.

VI.2. Analysis of Results

Across the different tests, we have demonstrated that networks with a CNN layer feeding into a self-regulating RNN layer performed better than networks that do not. This confirms our primary assumption that combining the two node types comprehends spatial and temporal features well. Our results are in-line with surveyed literature. Interestingly our testing also shows that when the input size is large (lots of sensor data) the simple CNN layer performs better than the simple RNN layer types. On the other hand, when the networks have small input data (i.e. one sensor) this advantage over the RNN layers diminishes.

The CNN-GRU architecture performed consistently best at recall, precision and accuracy against the validation data when using one sensor. Interestingly, the CNN-GRU architecture did not perform as well as the CNN-LSTM and CNN-BILSTM networks when using all sensors in the MHEALTH dataset. The CNN-LSTM and CNN-BILSTM networks see significant jumps of approximately 17 percent in validation data recall and precision when using all sensors versus using one sensor. This suggests a couple things. The first is that more complicated input benefits from a more complicated recurrent neural node. The GRU nodes performed best when the input size was only three: one accelerometer with measurements in the x, y, and z axis. When all sensors were included in the input, the input size ballooned to 23. It appears networks with the more complex memory management like LSTM and BILSTM understood this complex input better. This suggests that whereas a GRU is more efficient at training and requires less resources, as the input complexity increases, so too does the benefit of selecting a more complicated recurrent node. It should be noted that in some comparisons, the results of network performance are close and often within a percentage for accuracy. This difference becomes more pronounced when measuring recall and precision. For example, during the all sensors test, the CNN-GRU, CNN-LSTM, and CNN-BILSTM are all over 99.1 percent validation accuracy with a range of 0.4 percent. This range increases to 2.4 percent when comparing recall and precision.

The CNN and CNN-CNN networks performed very well against the training data but degraded more significantly than others when testing against the validation data. This suggests that the CNN only networks 'overfit' the training data more than other networks. Increasing the dropout for these networks may improve their performance.

Another trend across all tests is how poorly the RNN and CNN-RNN networks performed. One clear reason is that a simple recurrent node cannot handle noise. It does not have a way to regulate its recurrent weight, causing it to be unable to differentiate between which sections of the input ought to be remembered. The other issue is that its selection of recurrent weight is susceptible to the exploding/vanishing gradient problem. As discussed in previous sections, this can result in the recurrent value being significantly under-weighed or over-weighed. Looking at the training graphs in the Appendix, the RNN and CNN-RNN have significant drops in performance that then recover. This reflects the pattern of the vanishing gradient problem where honing in the recurrent weight becomes difficult. These findings are inline with the surveyed literature and the cited reasons for developing more complicated recurrent nodes such as LSTM and BILSTM. An interesting experiment could be to test CNN-RNN and RNN networks with different length inputs to demonstrate how the vanishing gradient problem becomes worse as the input size lengthens.

As shown in the training graphs in the Appendix, the CNN and CNN-CNN networks approach their performance limits faster than the other networks. For example when testing MHEALTH dataset using one accelerometer and no noise, the CNN networks are starting to asymptotically reach their performance limits within the first 200 epochs whereas networks such as the GRU, LSTM, and BILSTM take well over 600 epochs to begin reaching their asymptotic limits for fitting the data. The table below depicts the average real-time training length for each epoch when training on the MHEALTH dataset with one sensor and noise.

Training Time Per Epoch	
Model	**Training Time (milliseconds)**
CNN	2
RNN	5
CNN-CNN	3
GRU	1014
LSTM	1013
BILSTM	1016
CNN-RNN	1008
CNN-GRU	1017
CNN-LSTM	1015
CNN-BILSTM	2018

The real-time length for each epoch makes the training time difference between the CNN and CNN-CNN versus the recurrent networks more stark. Not only do the CNN networks approach their limits in fewer epochs but the CNN epoch length is approximately 500 times shorter than the recurrent epoch times. It should be noted that these tests were run using the Cuda drivers that allow Keras testing to run on the GPU. The GPU used for this testing is a NVIDIA GeForce GTX 1070. Understanding fully why there is such a significant performance difference would require more research into how Keras generates Cuda code to be run on the GPU. The likely cause is that the CNN and CNN-CNN networks can be more parallelized on the GPU. The recurrent networks are inherently linear as each input impacts the other. This likely means the code does not run well on a GPU. One interesting outlier is that the RNN network runs in 5ms while the CNN-RNN network runs in 1008ms. It is unclear why there is such a performance difference. It may be that the Cuda code generated by Keras for the CNN-RNN network degrades parallelization in some way.

Hyperparameter tuning showed that after selecting the 'right' types of nodes, the network can still be improved. Although our hyperparameter tuning is limited, it shows that the architecture can be altered to improve performance. In this paper's case, extending the size of the CNN and GRU layers as well as changing the kernel size from three to four in the CNN layers improved validation performance from 79.4 to 93.2. Additional testing could be done to more extensively explore how tuning alters a network's performance.

VI.3. Discussion

VI.3.1 Problems and issues arisen during the research

One challenge I found during testing was selecting the appropriate number of time steps for testing. I found that the techniques for extracting noisy and non-noisy data resulted in significantly different input sizes for training. Extracting non-noisy inputs required finding 100 contiguous timesteps of the same activity. In contrast, selecting noisy data did not require any constraints on the input, every sample could be used in the dataset. There was enough data in both datasets to train and test the networks but it is somewhat skewed to compare performance between noisy and non-noisy data because they are different sizes. In future work, it would be more accurate for comparing the two datasets if the number of samples were scaled to contain the same count and distribution of activity types.

Another challenge was comparing performance of the networks when discussing the WISDM and MHEALTH datasets. The main issue is that the sampling frequency for MHEALTH is 50Hz while WISDM is 20Hz. To try and make them more similar, both tests use a timeframe of 100 entries. The inherent problem of this is that WISDM ends up including 5 seconds of action while MHEALTH covers 2 seconds. This results in MHEALTH capturing a finer resolution sampling of the action. When comparing the average validation accuracy, precision, and recall for one accelerometer and noise (excluding RNN and CNN-RNN networks) the models trained using MHEALTH performed 4.3, 2.0, and 2.0 percent better on average than those trained using WISDM. I am suspicious this difference is partially attributable to the real-time input length difference between datasets. In future work, a more accurate comparison could be to select samples from MHEALTH at a 20Hz frequency: skipping entries to mimic a slower sampling rate. Another could be to interpolate and generate samples between the WISDM datasets entries. It would be interesting to see what the effects of interpolating entries between samples would do to training and if this is a reasonable approach.

VI.3.2 Limitations and constraints of the research (and results)

The initial project proposal considered comparing how the networks perform when the dataset was sensor data versus video data. Although classification of actions in a video is impressive, I quickly found that this comparison was flawed. First, there is not much insight to be gained by comparing the two types of datasets. Obviously, networks with CNN layers would perform significantly better when parsing videos. Additionally, there is not much value in comparing two categories of datasets that are so dissimilar. The valuable comparison is not in comparing how networks perform on two unrelated datasets but how networks perform in comparison to each other on the same dataset.

Another constraint is that the extent of any benchmarking is arbitrary. There are always more arrangements of network parameters whether it be activation functions selected, layer count, optimization function, kernel size, etc. Despite this, I attempted to alleviate the arbitrary nature by using networks that were relatively simple, consistent across architectures, and could be trained in a reasonable time frame. The goal of this paper is to compare how different layer types affect performance; therefore, the less complicated the network the less performance differences could be attributed to hyperparameters other than the node types.

The last significant constraint was the use of hyperparameter tuning. At the time of the project proposal, I was only cursurly familiar with hyperparameter tuning as I had just started to learn about it for the project proposal. The purpose for including hyperparameter tuning was to briefly explore the impact of tuning a selected architecture and demonstrate potential improvements. A more thorough benchmarking comparison could involve tuning each model in the tests to see which tuned model performs best. One flaw with this is that you only tune what you specify. A hyperparameter may significantly improve performance but this will be undiscovered unless explicitly targeted for tuning. Additionally, tuning takes a significantly long time. Tuning all models across the tests would be overwhelmingly cumbersome and significantly delay progress with long tuning times. An entire paper could be written looking at the effects of hyperparameter tuning on these models. We only looked at a couple arbitrary hyperparameters and did not even experiment with bayesian inference or random search tuning. It would be interesting to see what research there is on selecting hyperparameters to tune and if there is an 'optimal' or mathematical approach to selecting hyperparameters.

VII. Summary

Human Activity Recognition, HAR, is a classification problem of identifying the human action based on input. HAR is a popular topic with a wide range of applications such as health tracking, human-robot interaction, surveillance, elder care, and moderating content online. Input generally consists of either sequential two dimensional images (video) or sequential sensor data. HAR input contains a spatial dimension of understanding one time slice of input and also a temporal dimension of understanding how each time slice relates to each other.

HAR solutions were once dominated by traditional statistical approaches such as SVMs, decision trees, and clustering algorithms. Modern techniques have largely replaced these traditional machine learning techniques with deep learning. A commonly cited reason for this is because it removes much of the data prepping and domain expertise. For example, you do not need to be a subject domain expert for a CNN to extract features. It will do so automatically.

When selecting a deep learning architecture to use for Human Activity Recognition, it is valuable to have a comprehensive comparison of how each model will perform. Most papers surveyed state a deep CNN-LSTM model is the ideal model for HAR but do not fully quantify this assertion. For example, none of the papers surveyed include traditional RNN or GRU models in their comparisons. [Perez-Gamboa et. al. 2021 & Mutegeki and Han 2020] compare CNNs, LSTMs, and CNN-LSTM networks with various depths but do not discuss other types of RNNs. [Moradi et. al. 2022] also compares CNN, LSTM, CNN-LSTM and BILSTMs but does not

include CNN-BILSTM networks. Additionally, the surveyed literature often limits the metrics of performance and only focuses on accuracy. This paper addresses that lack of comparison, and provides a benchmarking of the commonly used deep learning architectures discussed in the surveyed literature. This paper provides more subtlety to the performance comparison and shows that selecting the 'best' architecture is dependent on what metrics are most valued. This paper includes a benchmarking of architectures that are commonly mentioned in the surveyed literature:

- Traditional RNN
- CNN
- GRU
- LSTM
- BILSTM
- CNN-RNN
- CNN-GRU
- CNN-LSTM
- CNN-BILSTM

This paper implements these models using the WISDM and MHEALTH sensor datasets. We also compare the performance of these models using accuracy, precision, recall, and training time. Hyperparameter tuning is then applied to the best performing model to measure how the tuning improves performance and provide insight into which features are most impactful to performance.

Across the different tests, we have demonstrated that networks with a CNN layer feeding into a self-regulating RNN layer performed better than networks that do not. This confirms our assumption that combining the two node types comprehends spatial and temporal features well. Interestingly our testing also shows that when the input size is large (lots of sensor data) the simple CNN layer performs better than the simple RNN layer types. On the other hand, when the networks have small input data (i.e. one sensor) this advantage over the RNN layers diminishes.

The CNN-GRU architecture performed consistently best at recall, precision and accuracy against the validation data when using one sensor. Interestingly, the CNN-GRU architecture did not perform as well as the CNN-LSTM and CNN-BILSTM networks when using all sensors in the MHEALTH dataset. The CNN-LSTM and CNN-BILSTM networks see significant jumps of approximately 17 percent in validation data recall and precision when using all sensors versus using one sensor. This suggests a couple things. The first is that more complicated input benefits from a more complicated recurrent neural node. The GRU nodes performed best when the input size was only three: one accelerometer with measurements in the x, y, and z axis. When all sensors were included in the input, the input size ballooned to 23. It appears networks with the more complex memory management like LSTM and BILSTM understood this complex input better.

The CNN and CNN-CNN networks performed very well against the training data but degraded more significantly than others when testing against the validation data. This suggests that the CNN only networks 'overfit' the training data more than other networks. Increasing the dropout for these networks may improve their performance. Another trend across all tests is how poorly the RNN and CNN-RNN networks performed. One clear reason is that a simple recurrent node cannot handle noise.

As shown in the training graphs in the Appendix, the CNN and CNN-CNN networks approach their performance limits faster than the other networks. For example when testing MHEALTH dataset using one accelerometer and no noise, the CNN networks are starting to asymptotically reach their performance limits within the first 200 epochs whereas networks such as the GRU, LSTM, and BILSTM take well over 600 epochs to begin reaching their asymptotic limits for fitting the data. Not only do the CNN networks approach their limits in fewer epochs but the CNN epoch length is approximately 500 times shorter than the recurrent epoch times. Understanding fully why there is such a significant performance difference would require more research into how Keras generates Cuda code to be run on the GPU. The likely cause is that the CNN and CNN-CNN networks can be more parallelized on the GPU.

Hyperparameter tuning showed that after selecting the 'right' types of nodes, the network can still be improved. Although our hyperparameter tuning is limited, it shows that the architecture can be altered to improve performance. In this paper's case, extending the size of the CNN and GRU layers as well as changing the kernel size from three to four in the CNN layers improved validation performance from 79.4 to 93.2.

Future work could be extended to test how sampling frequency impacts network performance. Additionally, more testing could be applied to traditional RNN networks to quantify how their performance degrades as the input size increases. An interesting finding from measuring the training time found that the RNN network trained an epoch in 5ms while the CNN-RNN network trained in 1008ms. It is still unclear why these two networks have such significantly different training times. These tests were performed on a GPU using Cuda code generated by Keras. Further work could investigate why there is such a significant discrepancy and whether it is related to how the Cuda code is generated. Finally, this paper briefly touched on hyperparameter tuning. Future research could further explore how networks perform after being tuned and what are the best practices for selecting parameters for tuning.

References

A. B. Gumelar, E. M. Yuniarno, D. P. Adi, R. Setiawan, I. Sugiarto and M. H. Purnomo, "Transformer-CNN Automatic Hyperparameter Tuning for Speech Emotion Recognition," *Proceedings of 2022 IEEE International Conference on Imaging Systems and Techniques (IST)*, Kaohsiung, Taiwan, 2022, pp. 1-6.

B. Lee, J. Kim and S. -U. Jung, "Light-weighted Network based Human Pose Estimation for Mobile AR Service," *Proceedings of 2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2020, pp. 1609-1612.

B. Moradi, M. Aghapour and A. Shirbandi, "Compare of Machine Learning and Deep Learning Approaches for Human Activity Recognition," *Proceedings of 2022 30th International Conference on Electrical Engineering (ICEE)*, Tehran, Iran, 2022, pp. 592-596.

C. Indhumathy, "An Introduction to Support Vector Machine," *https://towardsdatascience.com/an-introduction-to-support-vector-machine-3f353241303b*. (as of February 12, 2023).

C. Shiranthika, N. Premakumara, H. -L. Chiu, H. Samani, C. Shyalika and C. -Y. Yang, "Human Activity Recognition Using CNN & LSTM," *Proceedings of 2020 5th International Conference on Information Technology Research (ICITR)*, Moratuwa, Sri Lanka, 2020, pp. 1-6.

H. Qian, Y. Mao, Honghua Wang and Z. Wang, "On video-based human action classification by SVM decision tree," *Proceedings of 2010 8th World Congress on Intelligent Control and Automation*, Jinan, China, 2010, pp. 385-390.

I. N. Yulita, E. Paulus, A. Sholahuddin and D. Novita, "AdaBoost Support Vector Machine Method for Human Activity Recognition," *Proceedings of 2021 International Conference on Artificial Intelligence and Big Data Analytics*, Bandung, Indonesia, 2021, pp. 1-4.

J. Brownlee, "How to choose an activation function for deep learning," *https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/* (as of February 12, 2023).

N. Ackermann, "Human activity recognition (HAR) tutorial with Keras and core ML (part 1)" *https://towardsdatascience.com/human-activity-recognition-har-tutorial-with-keras-and-core-ml-part-1-8c05e365dfa0* (as of March 22, 2023).

R. Dolphin, "LSTM Networks | A Detailed Explanation." *https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9* (as of February 12, 2023).

R. Mutegeki and D. S. Han, "A CNN-LSTM Approach to Human Activity Recognition," *Proceedings of 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Fukuoka, Japan, 2020, pp. 362-366.

R. Poojary and A. Pai, "Comparative Study of Model Optimization Techniques in Fine-Tuned CNN Models," *Proceedings of 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, 2019, pp. 1-4.

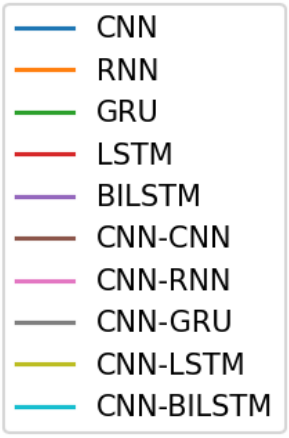
S. Joshi, J. A. Owens, S. Shah and T. Munasinghe, "Analysis of Preprocessing Techniques, Keras Tuner, and Transfer Learning on Cloud Street image data," *Proceedings of 2021 IEEE International Conference on Big Data (Big Data)*, Orlando, FL, USA, 2021, pp. 4165-4168

S. Perez-Gamboa, Q. Sun and Y. Zhang, "Improved Sensor Based Human Activity Recognition via Hybrid Convolutional and Recurrent Neural Networks," *Proceedings of 2021 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, Kailua-Kona, HI, USA, 2021, pp. 1-4.

Y. Wei and L. Jiang, "Human action recognition based on convolutional neural network," *Proceedings of 2021 2nd International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI),* Shenyang, China, 2021, pp. 194-197.

IIX. Appendix

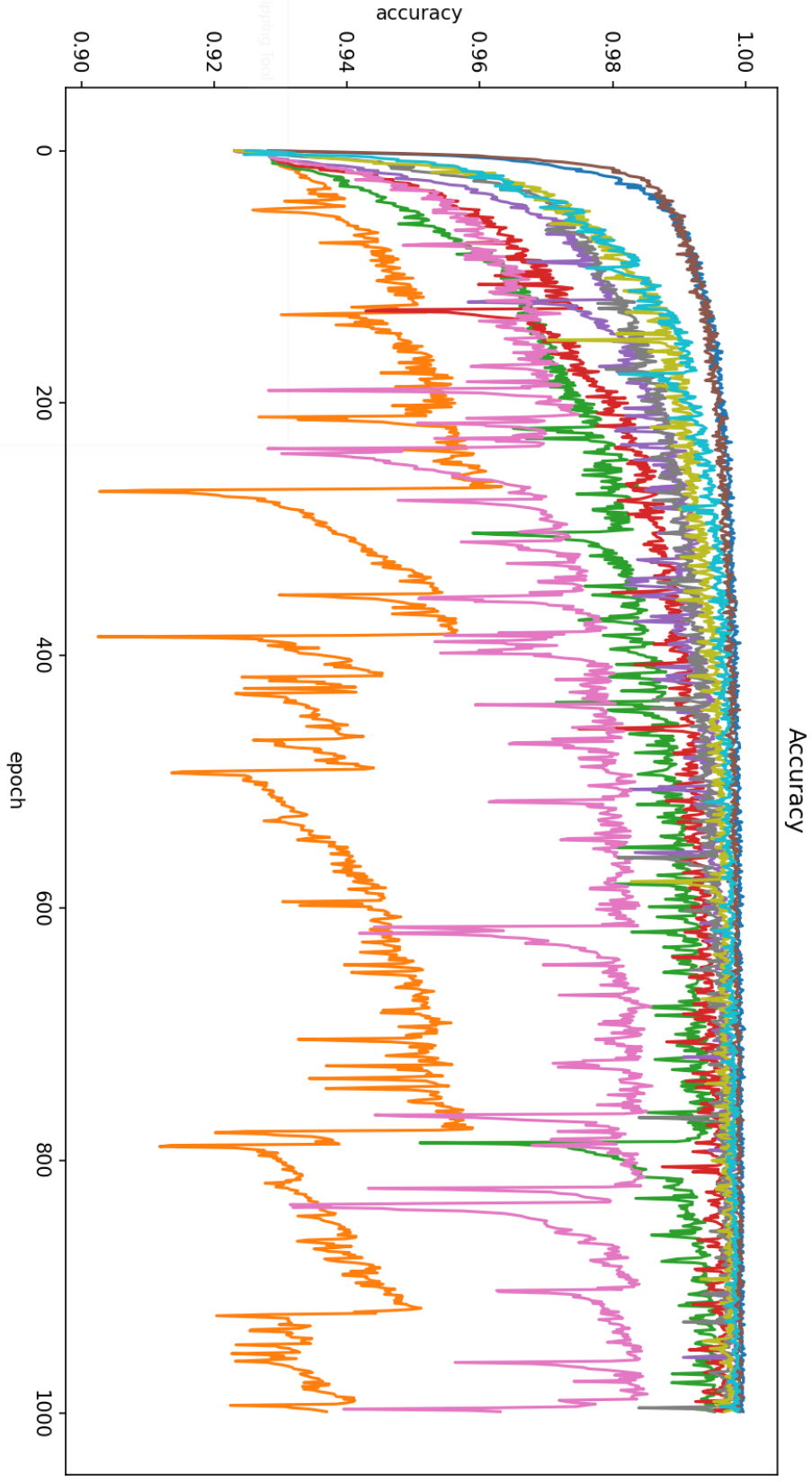
Listed below are the training results for the different tests. To make the graphs more legible, the legend for all line graphs is listed below:

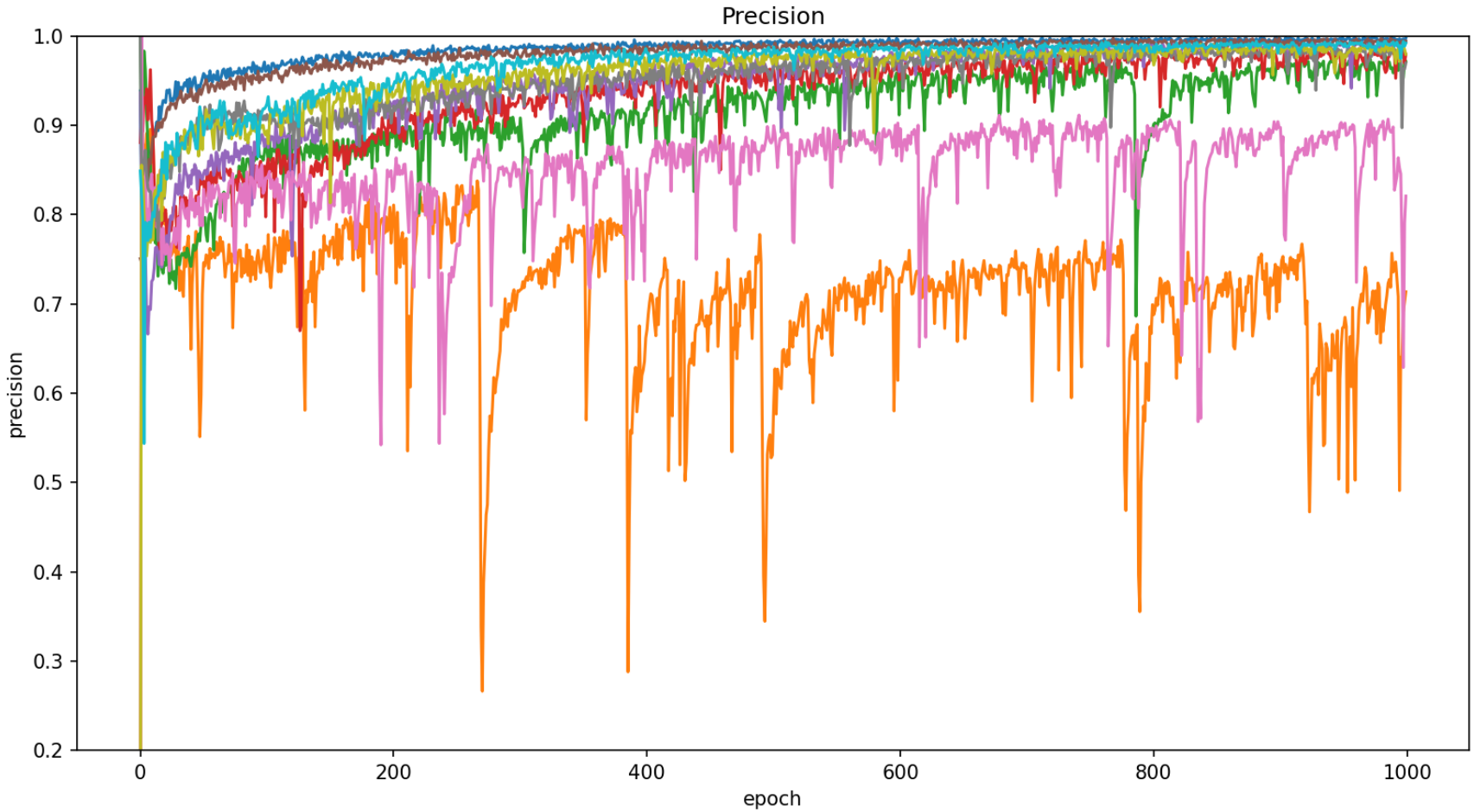


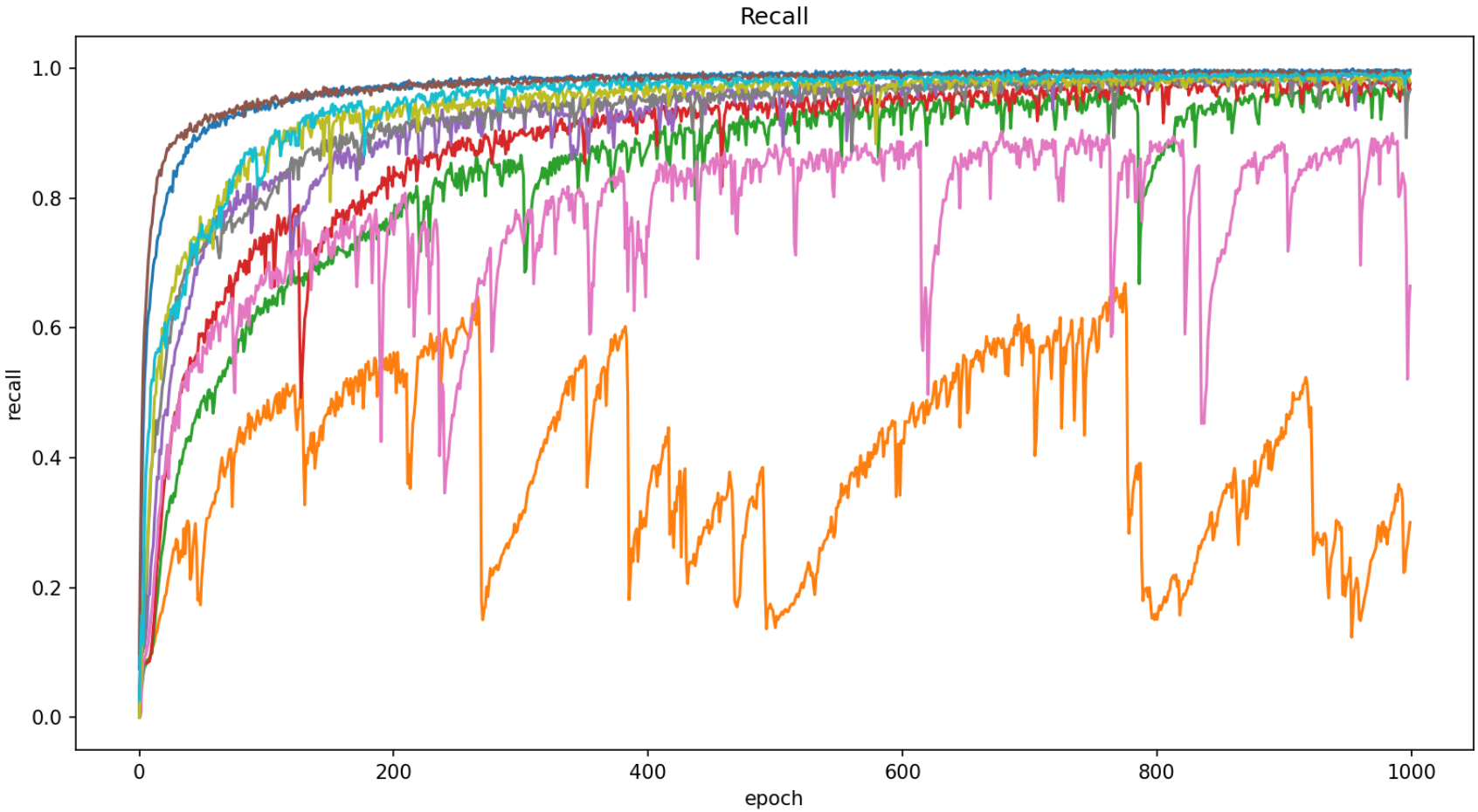
The MHEALTH dataset's twelve activities are difficult to read in the generated confusion matrices. Their values from left to right and top to bottom are as follows:

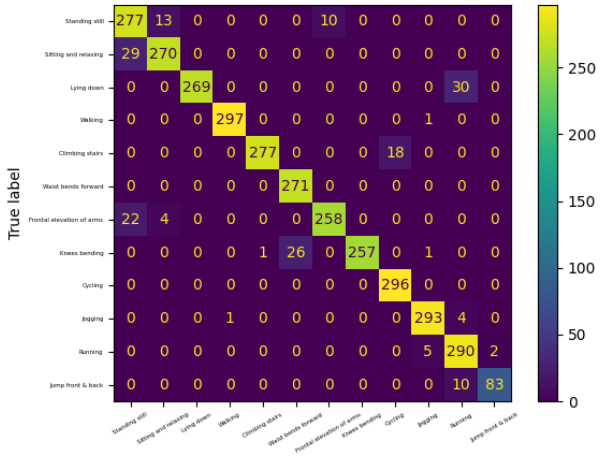
- Stand still
- Sitting
- Lying
- Walking
- Climb stairs
- Waist bend
- Elevate arms
- Bend Knees
- Cycling
- Jogging
- Running
- Jumping

MHEALTH Dataset Using One Accelerometer and No Noise:

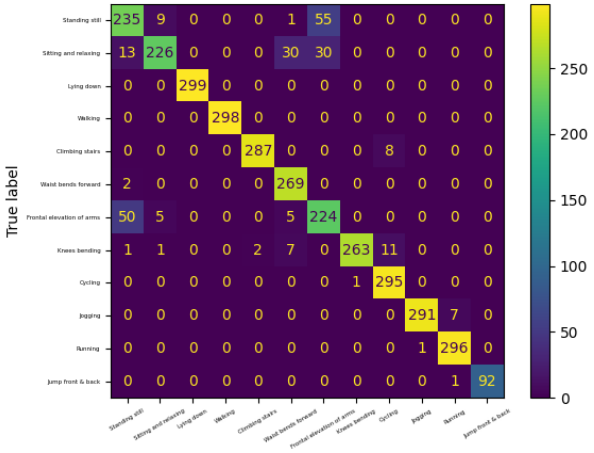



Precision
precision
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000

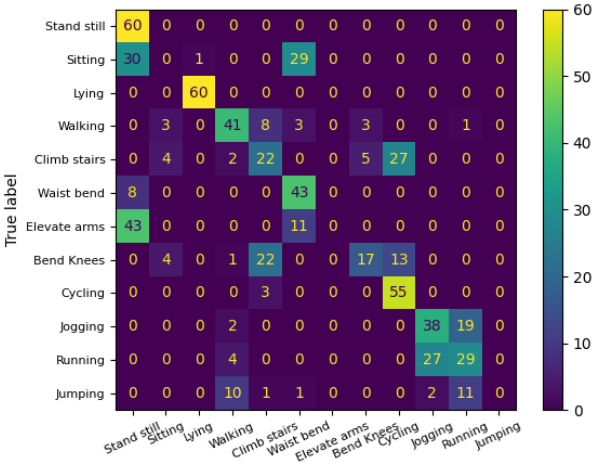

Recall
recall
epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000



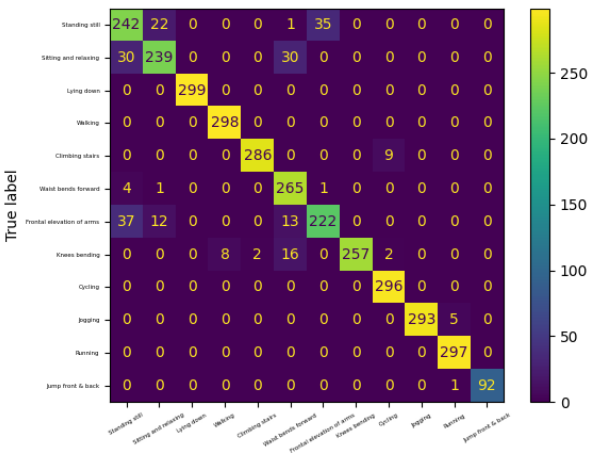
CNN



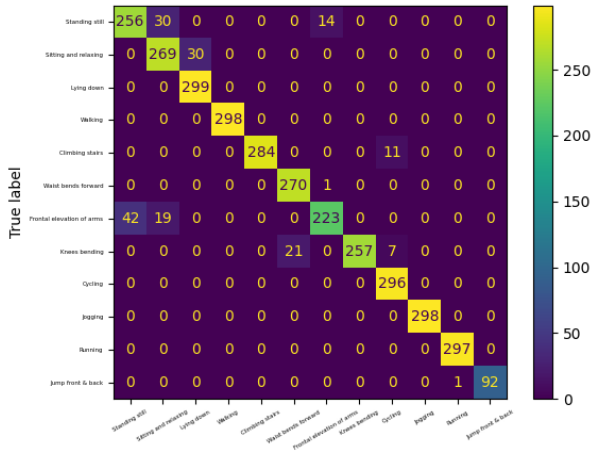
GRU



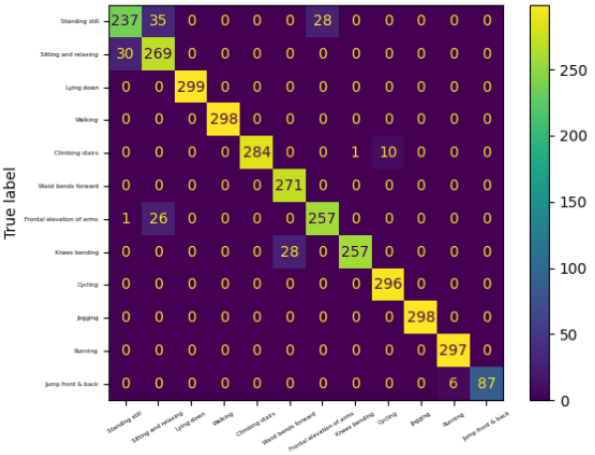
RNN



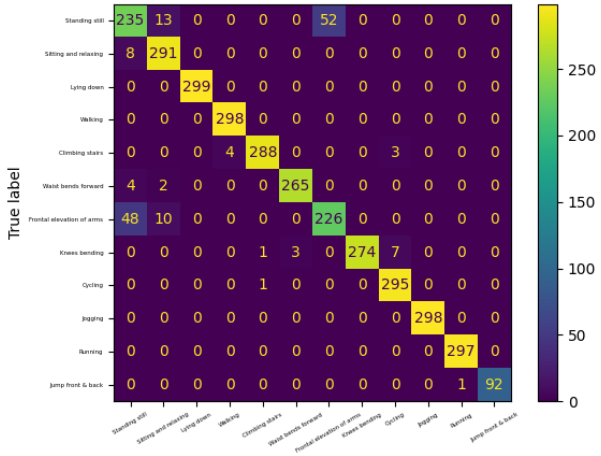
LSTM



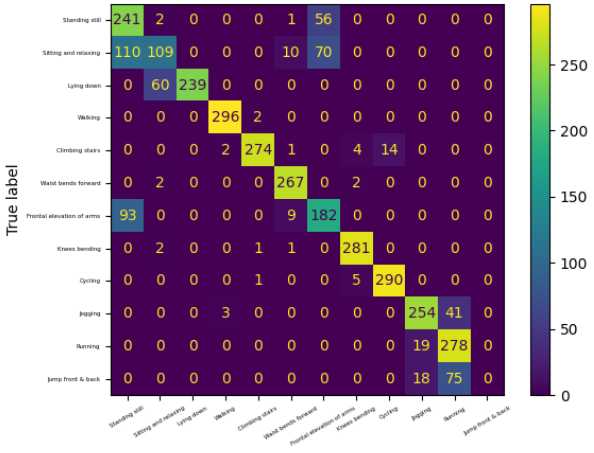
BILSTM



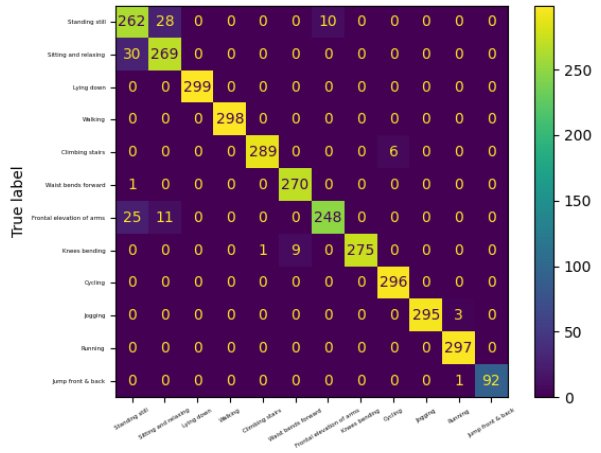
CNN-CNN



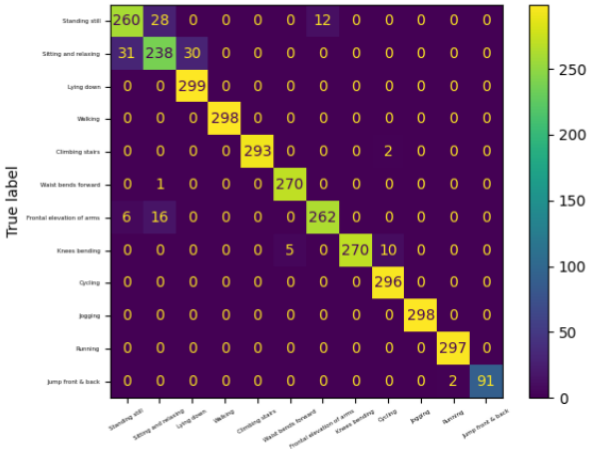
CNN-GRU



CNN-RNN

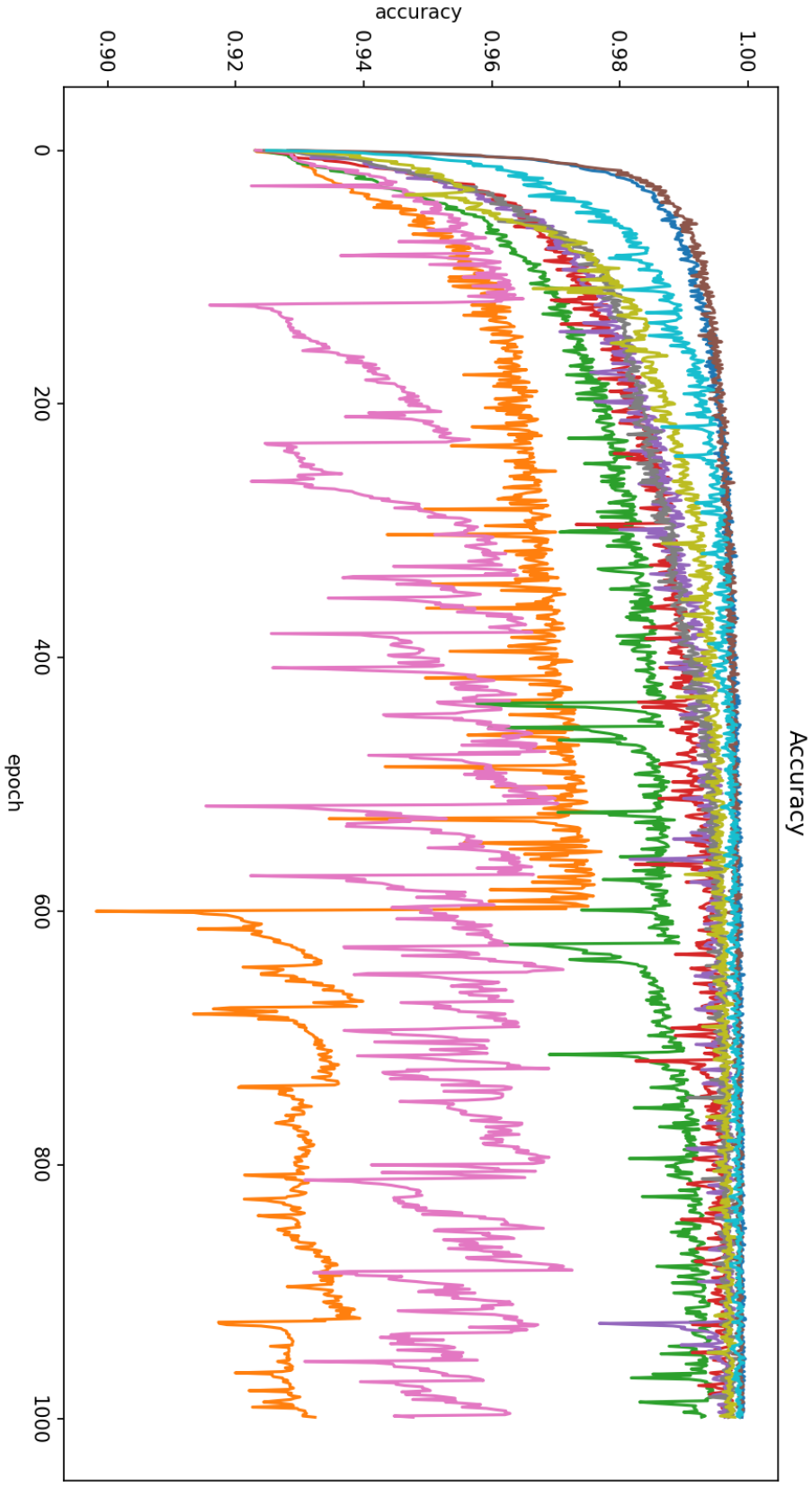


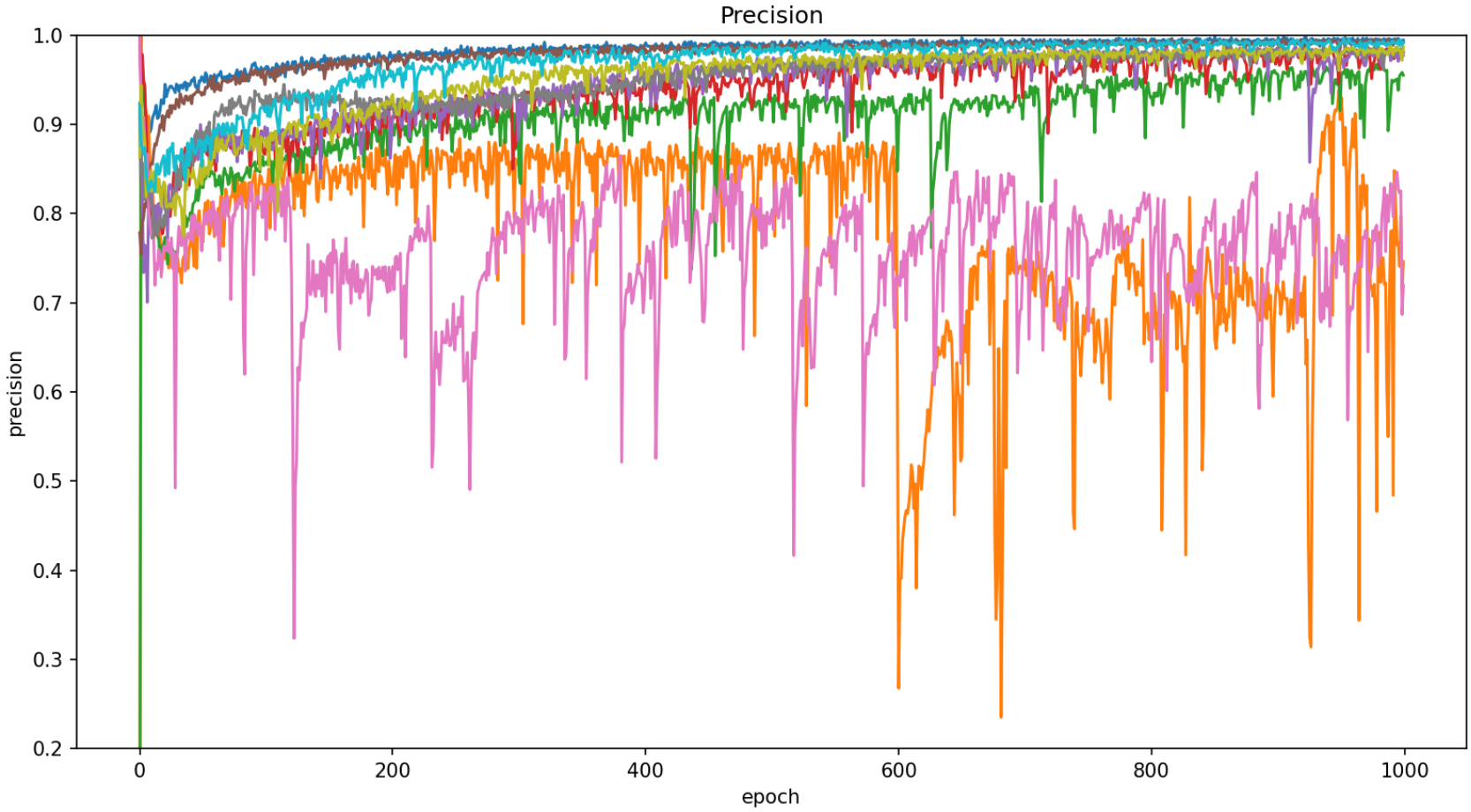
CNN-LSTM

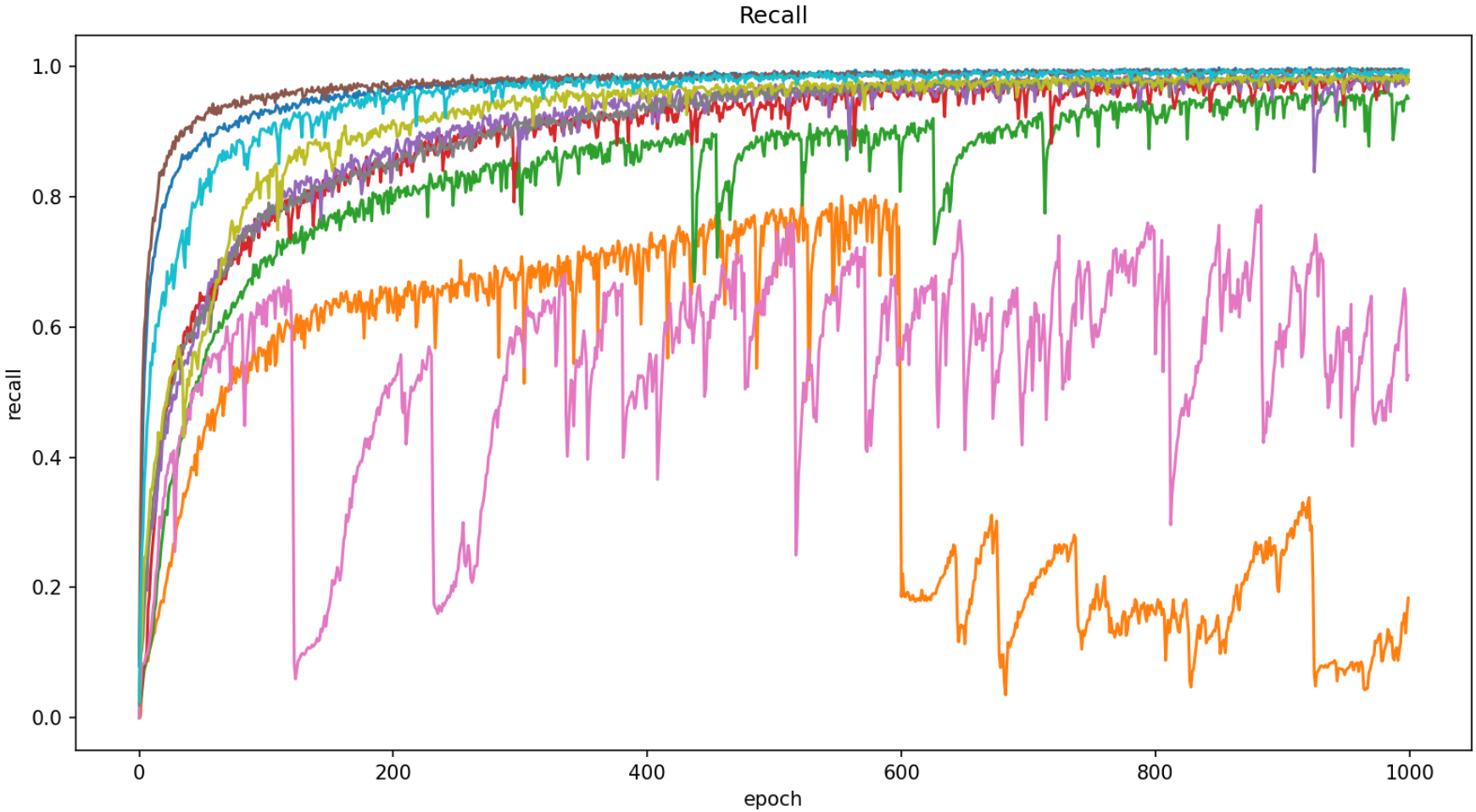


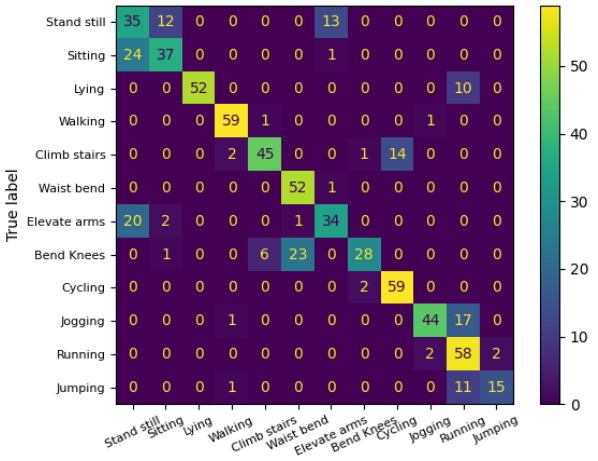
CNN-BILSTM

MHEALTH Dataset Using One Accelerometer and Noise:

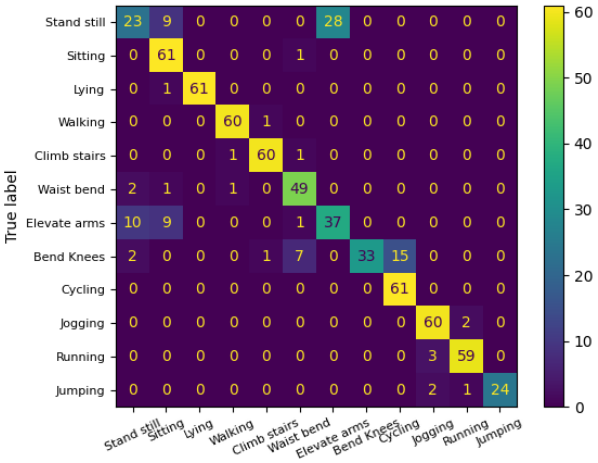



Precision
precision
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000

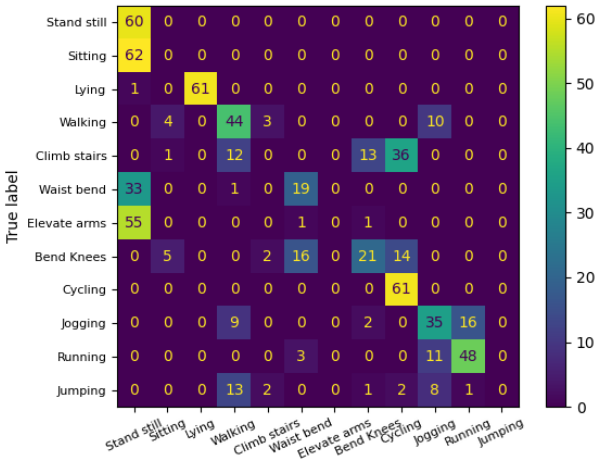

Recall
recall
epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000



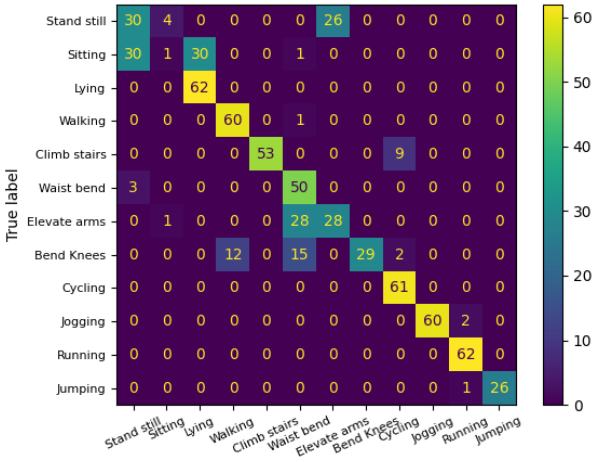
CNN



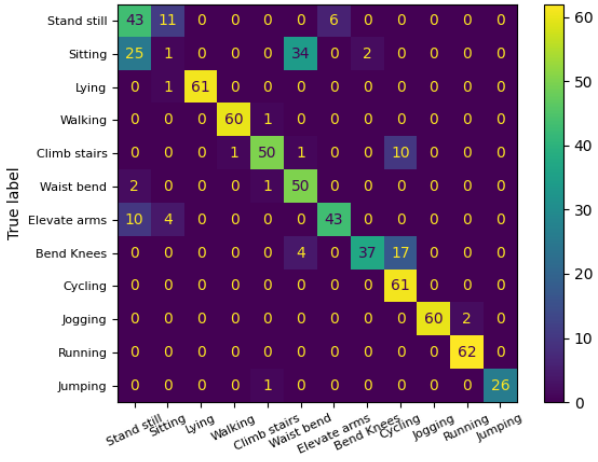
GRU



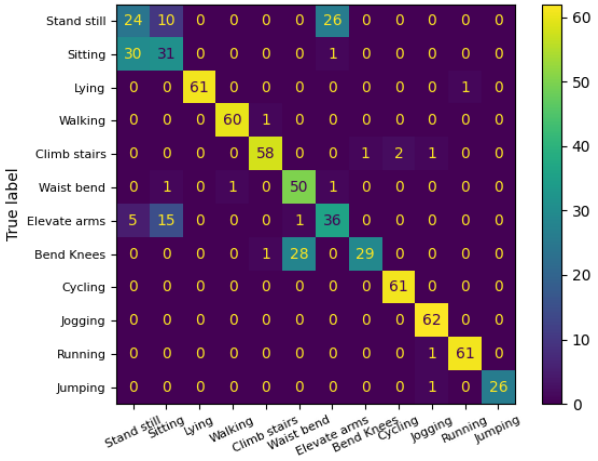
RNN



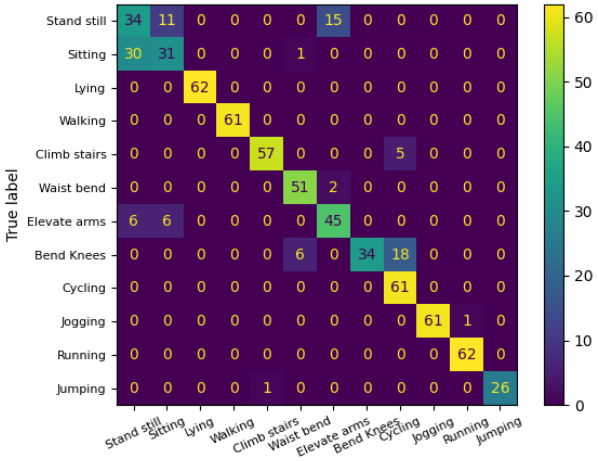
LSTM



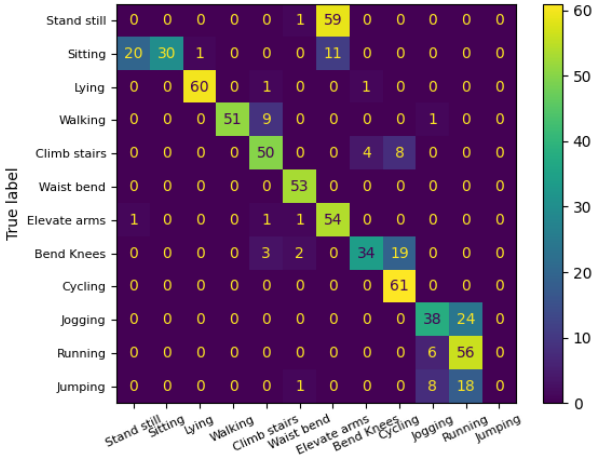
BILSTM



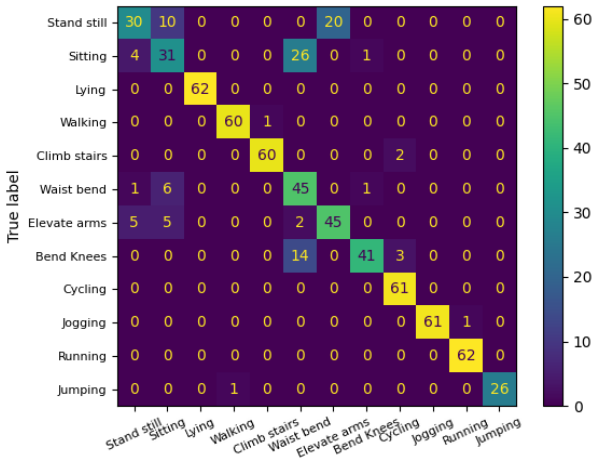
CNN-CNN



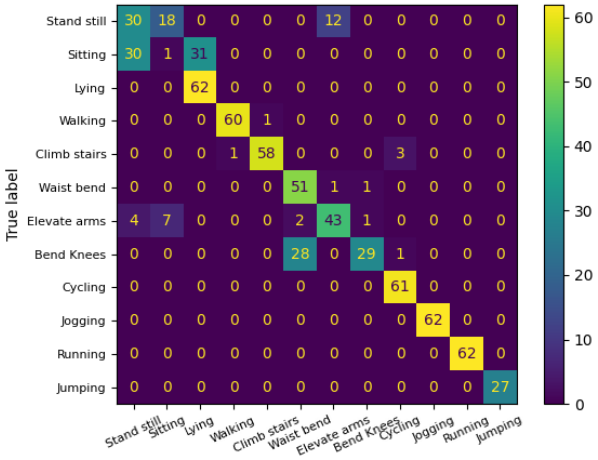
CNN-GRU



CNN-RNN

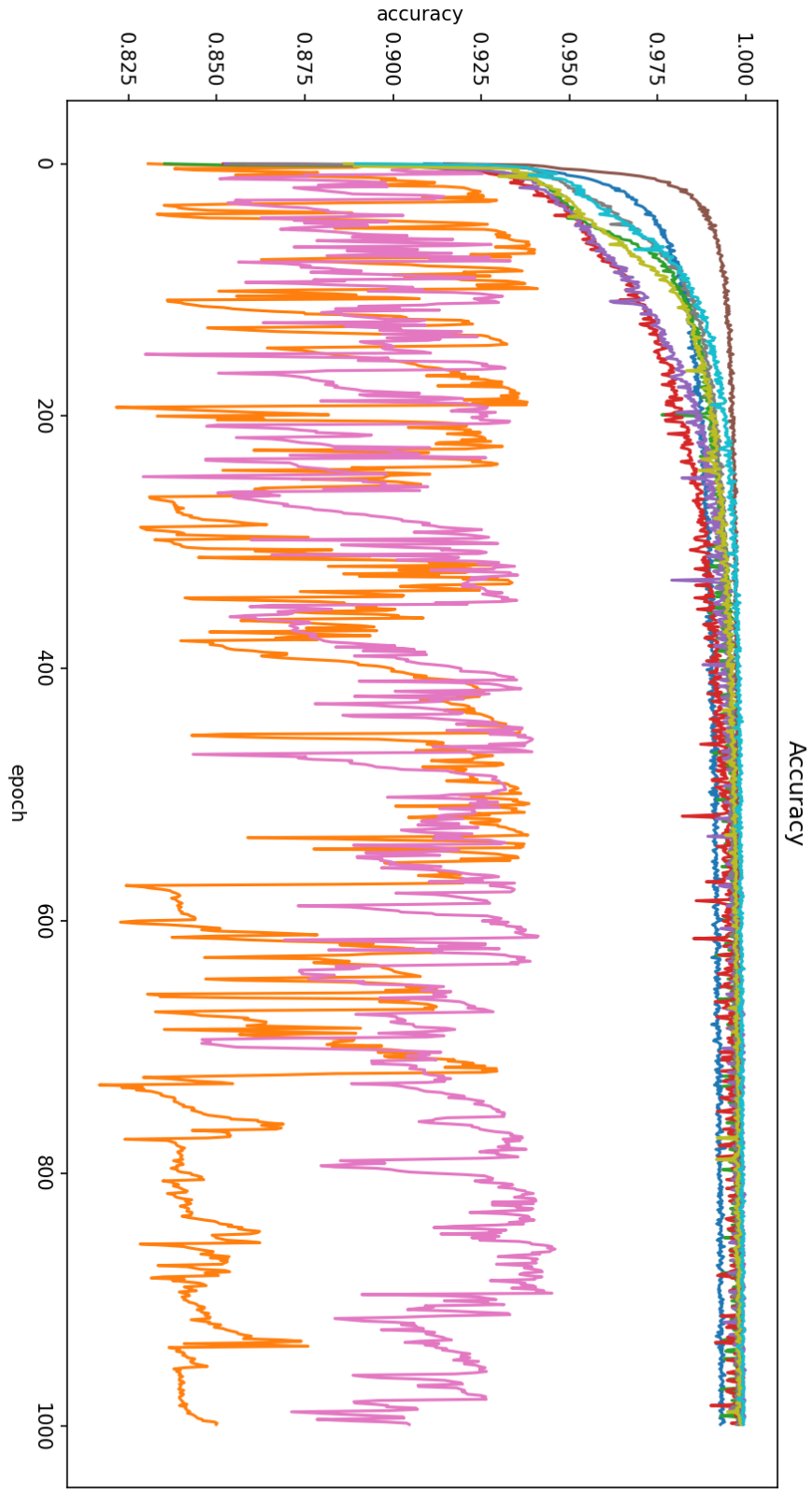


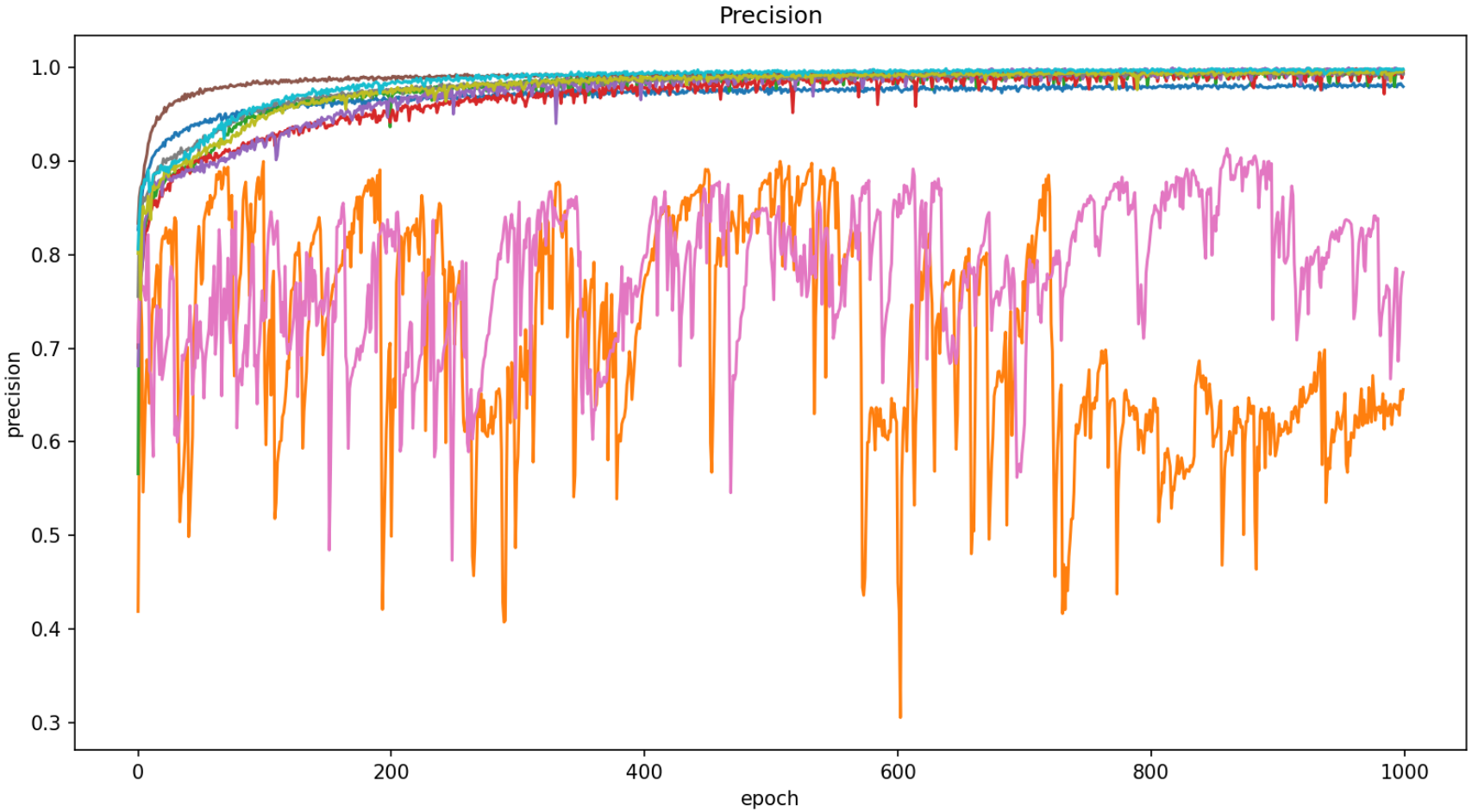
CNN-LSTM

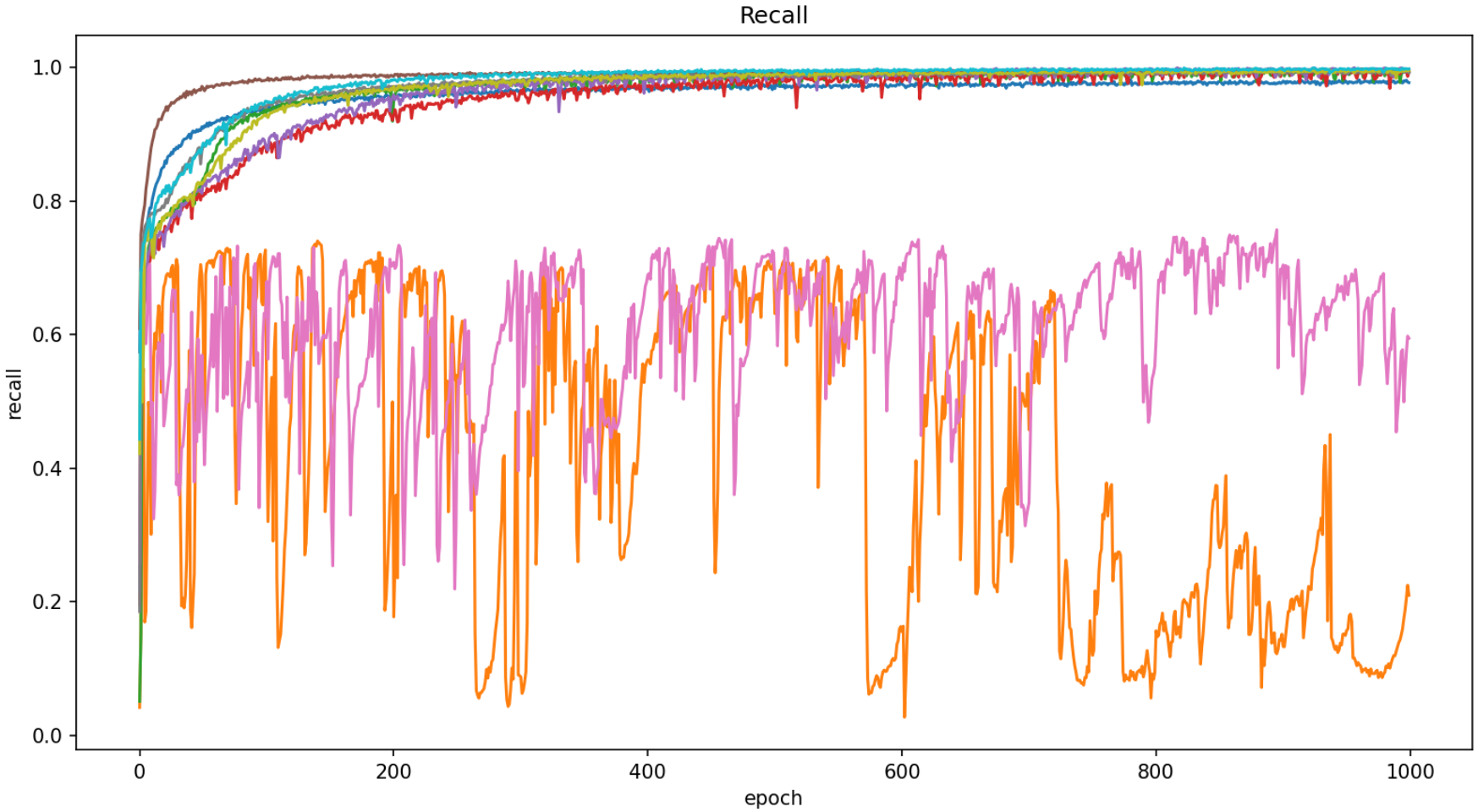


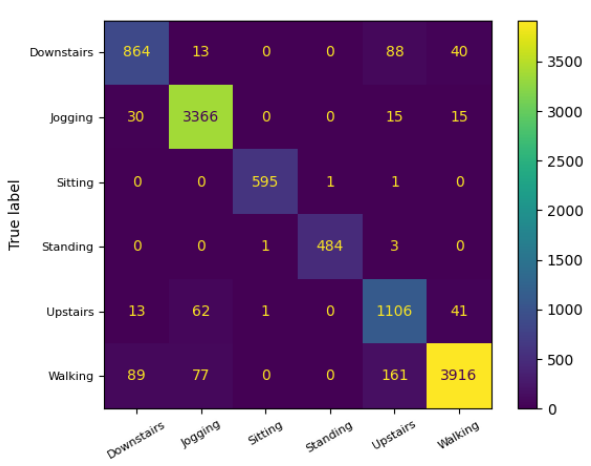
CNN-BILSTM

WISDM Dataset Using One Gyroscope and Noise:

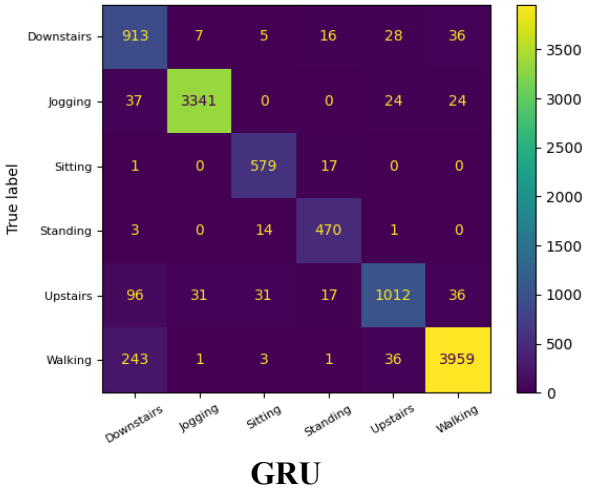



Precision
precision
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
200
400
600
800
1000

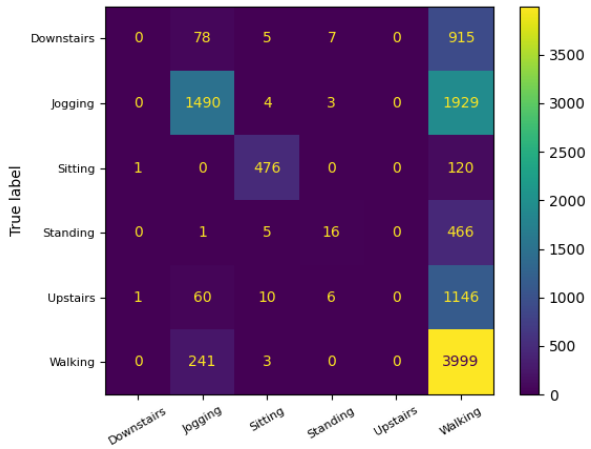

Recall
recall
epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000



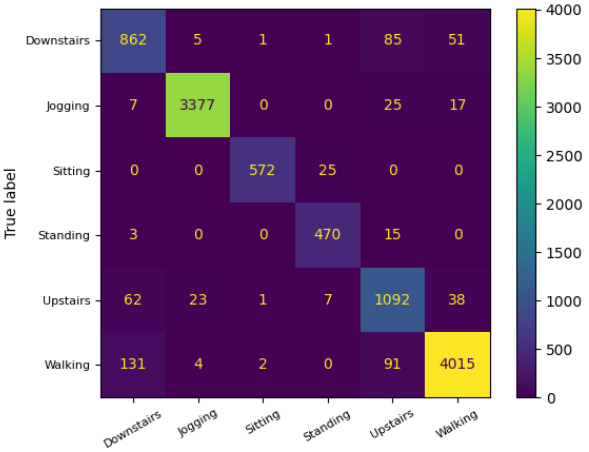
CNN



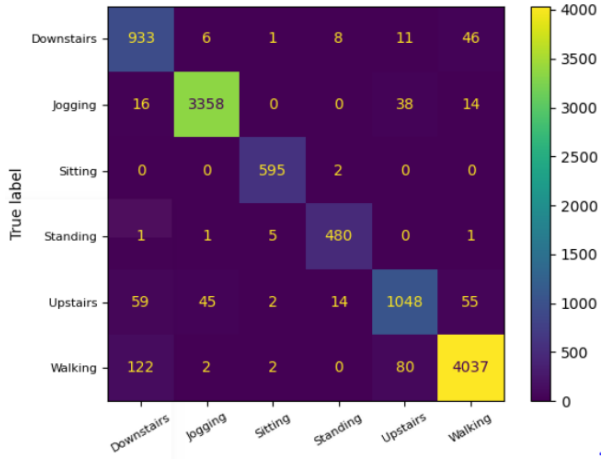
GRU



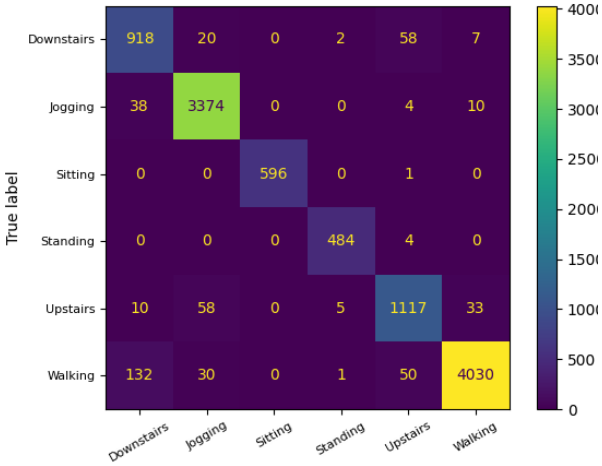
RNN



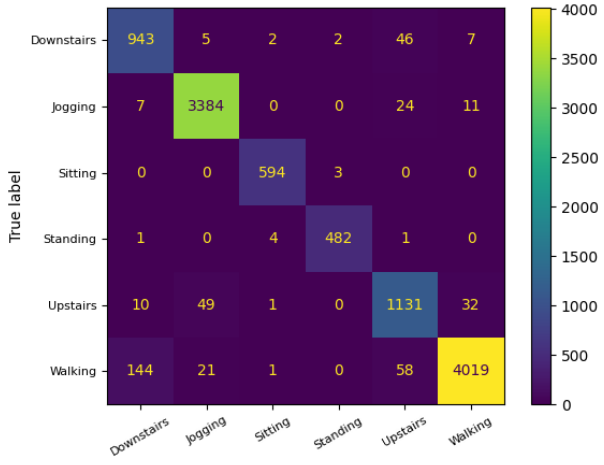
LSTM



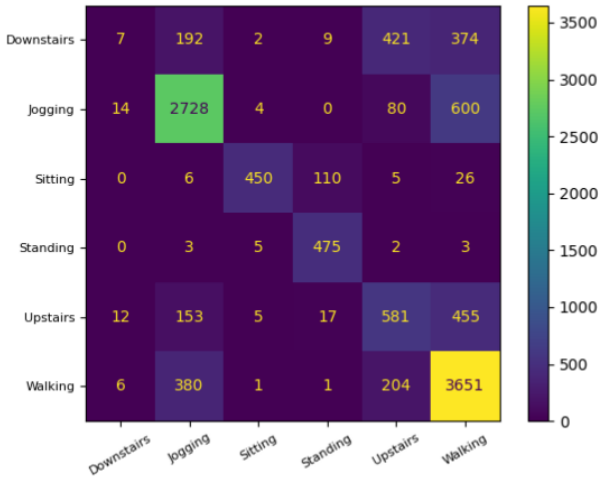
BILSTM



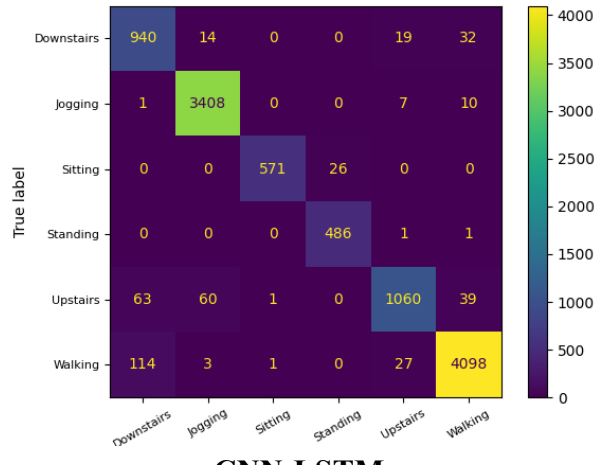
CNN-CNN



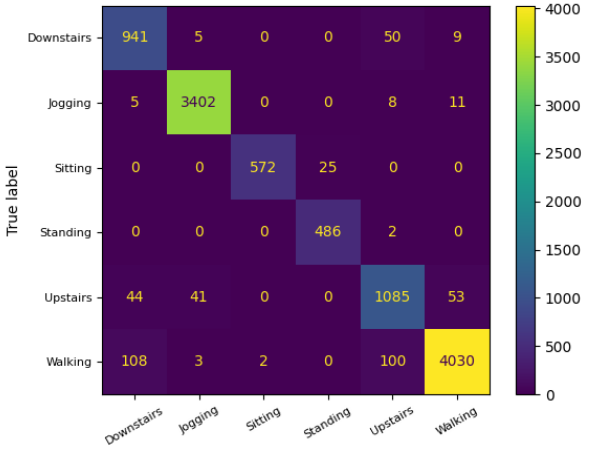
CNN-GRU



CNN-RNN

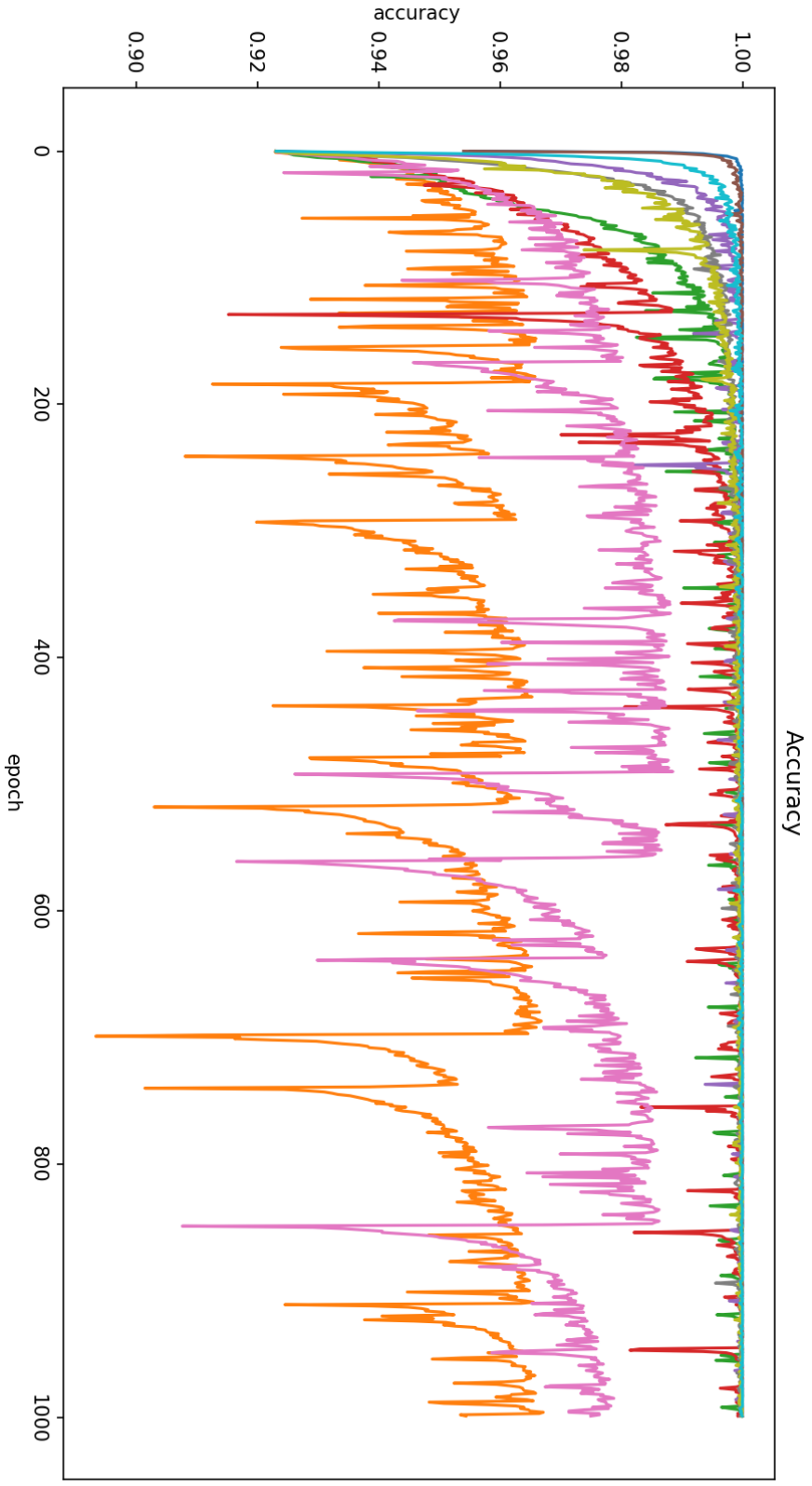


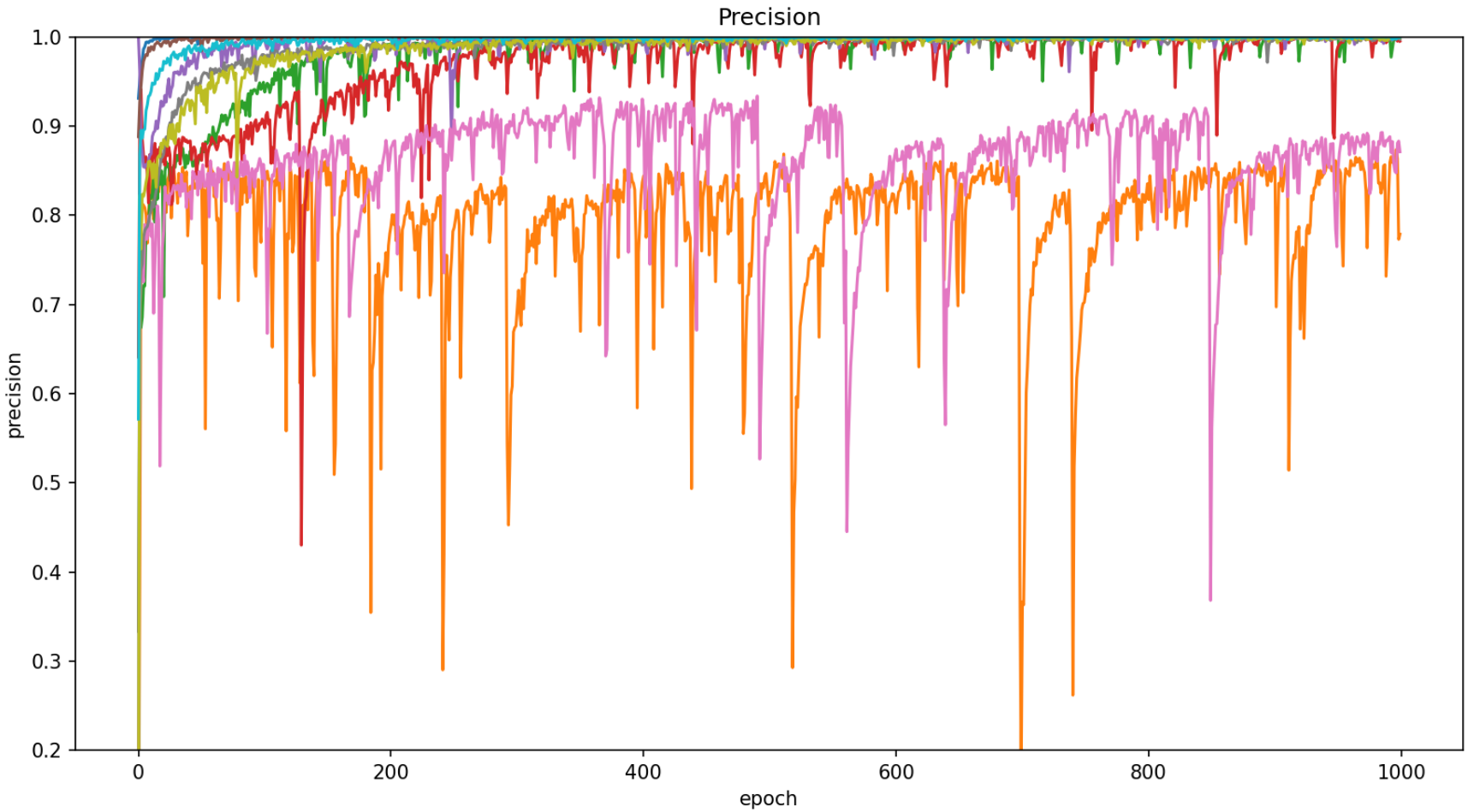
CNN-LSTM

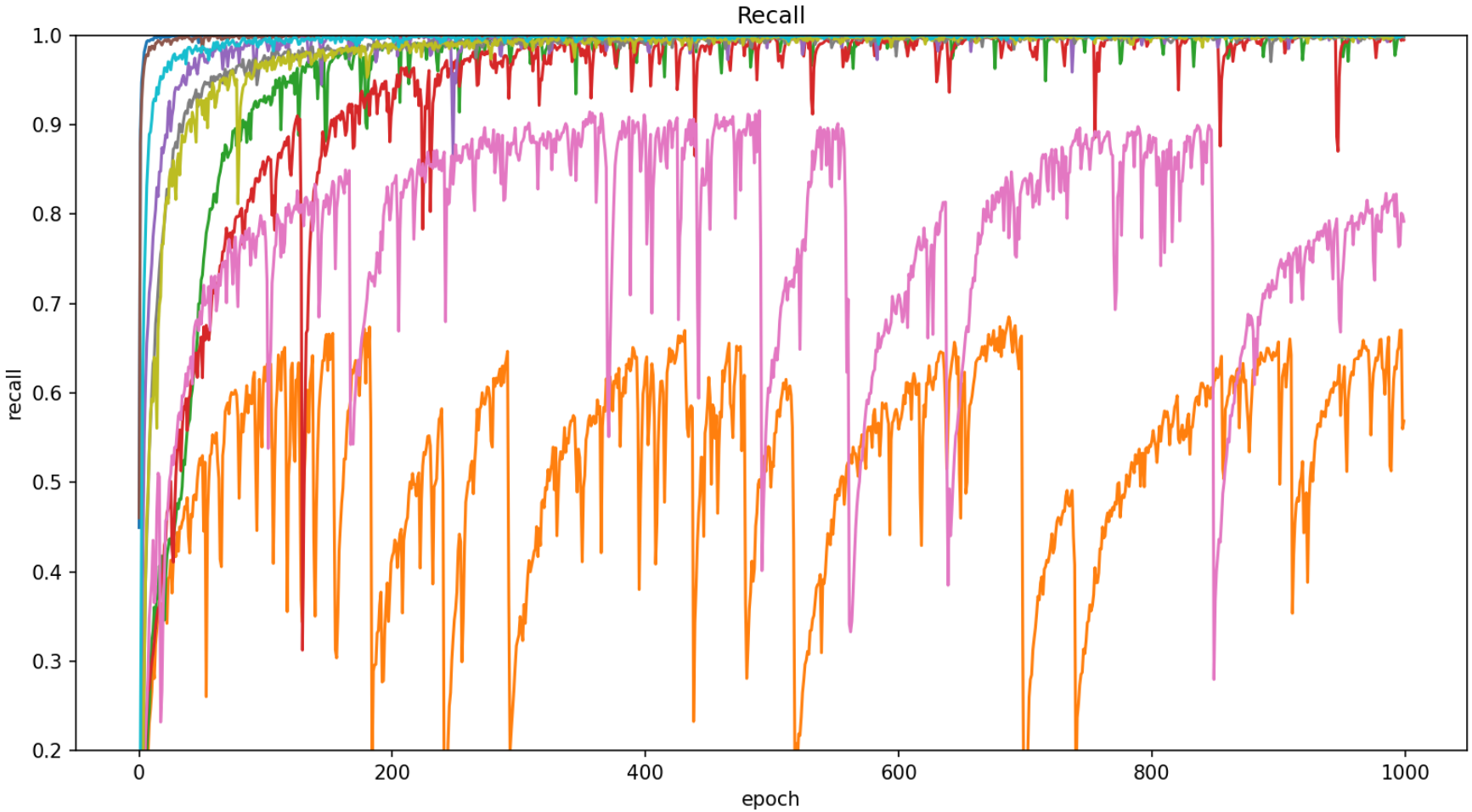


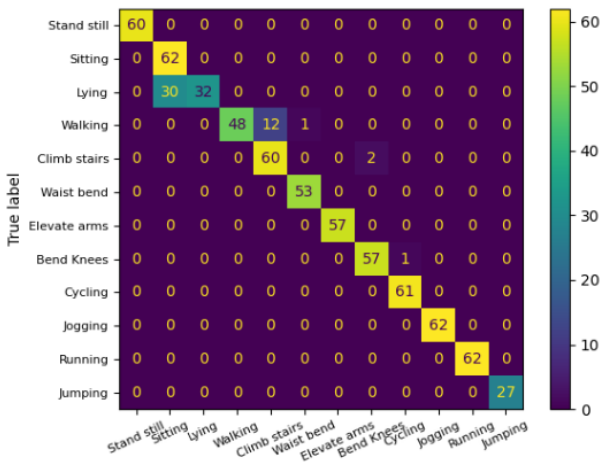
CNN-BILSTM

MHEALTH Dataset Using All Sensors and Noise:

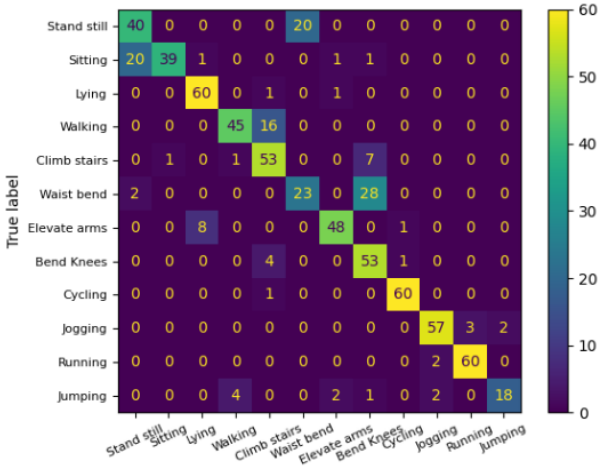



Precision
precision
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000

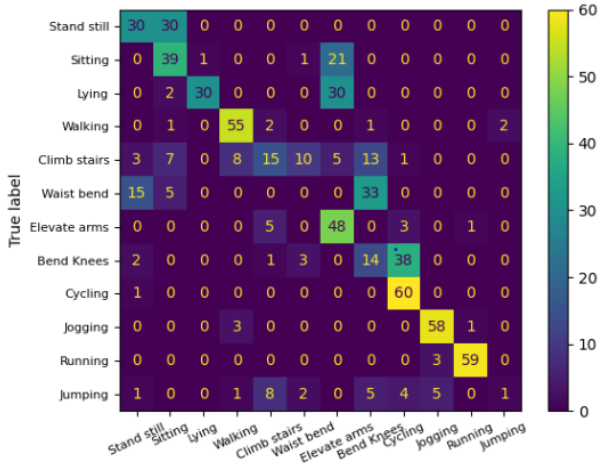

Recall
recall
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000



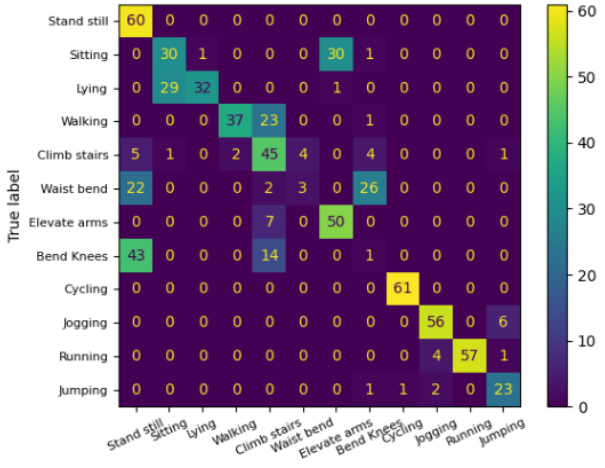
CNN



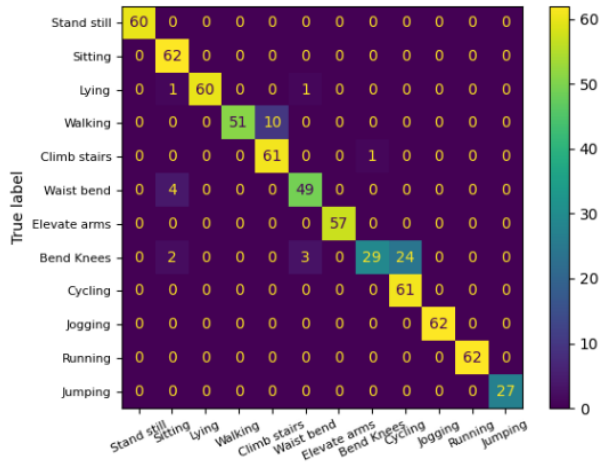
GRU



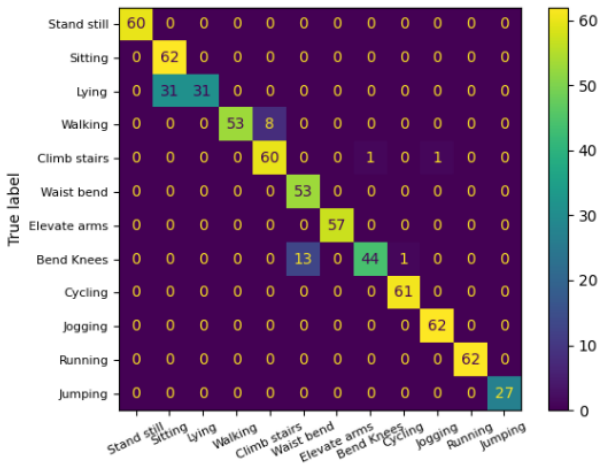
RNN



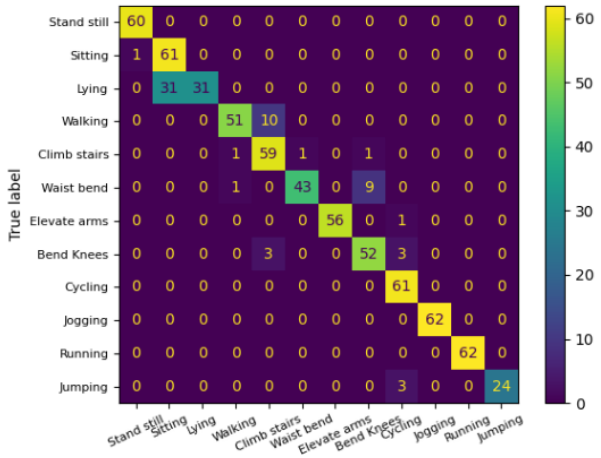
LSTM



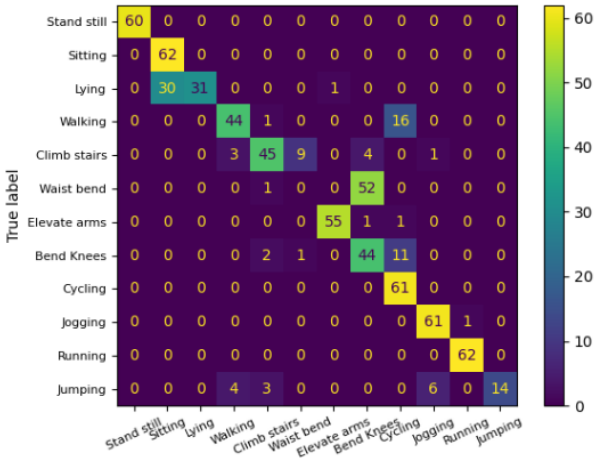
BILSTM



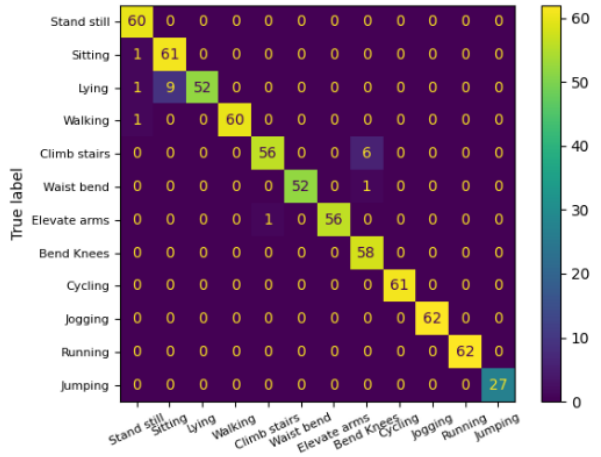
CNN-CNN



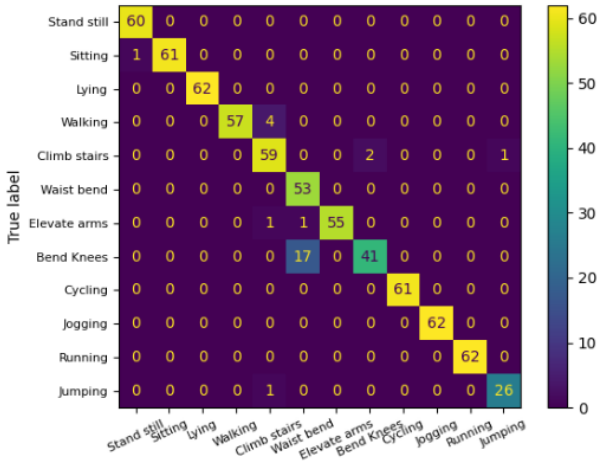
CNN-GRU



CNN-RNN



CNN-LSTM



CNN-BILSTM