

**DUBLIN CITY UNIVERSITY**

**SCHOOL OF ELECTRONIC ENGINEERING**

**A Project Template**

**and Report Writing Guide**

**Mark Kane**

August 2019

BACHELOR OF ENGINEERING

IN

ELECTRONIC SYSTEMS

Supervised by Prof. N. O’Connor

# Acknowledgements

# Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at <https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf> and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=448779.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

Table of Contents

[Acknowledgements ii](#_Toc5495531)

[Declaration ii](#_Toc5495532)

[Abstract iii](#_Toc5495533)

[Chapter 1 - Introduction 1](#_Toc5495534)

[1.1 Human Activity Recognition 1](#_Toc5495535)

[1.1.1 Applications of HAR 1](#_Toc5495536)

[1.2 Types of HAR 2](#_Toc5495537)

[1.3 HAR: Machine Learning vs Deep Learning 3](#_Toc5495538)

[Chapter 2 - Technical Background & Literature Review 5](#_Toc5495539)

[2.1 Sensing Technologies 5](#_Toc5495540)

[2.1.1 RGB Video 5](#_Toc5495541)

[2.1.2 Depth Sensor 5](#_Toc5495542)

[2.1.3 Wearable Sensor 6](#_Toc5495543)

[2.3 Sensor Modalities 6](#_Toc5495544)

[2.4 CNNs and HAR 8](#_Toc5495545)

[2.4.1 Input Adaptation for CNNs 9](#_Toc5495546)

[2.4.2 Pooling 9](#_Toc5495547)

[2.4.3 Weight Sharing 9](#_Toc5495548)

[2.6 Common Approaches to HAR 9](#_Toc5495549)

[2.7 Project Outline 12](#_Toc5495550)

[2.7 Modality Transformation 13](#_Toc5495551)

[Chapter 3 - Design of Solutions 14](#_Toc5495552)

[3.1 The Dataset – Pamap2 Dataset 14](#_Toc5495553)

[3.1.1 Hardware Setup 14](#_Toc5495554)

[3.1.2 Subject Information 14](#_Toc5495555)

[3.1.3 Data Collection Activity Protocol 15](#_Toc5495556)

[3.1.4 The Collected Data 16](#_Toc5495557)

[3.1.5 Data Format 17](#_Toc5495558)

[3.2 Baseline Approach 18](#_Toc5495559)

[3.2.1 Sliding Window 18](#_Toc5495560)

[3.2.2 Temporal Convolution and Pooling 20](#_Toc5495561)

[3.2.2 The Architecture 20](#_Toc5495562)

[3.3 Experiment Design 22](#_Toc5495563)

[3.3.1 Machine Vision Approach 1 (called 2 in code) 24](#_Toc5495564)

[3.3.2 Machine Vision Approach 2 (called 1 in code) 26](#_Toc5495565)

[3.3.3 Machine Vision Approach 3 26](#_Toc5495566)

[Chapter 4- Implementation and Testing 28](#_Toc5495567)

[4.1 Dataset Design Protocol 28](#_Toc5495568)

[4.2 Data Pre-Processing 28](#_Toc5495569)

[4.3 Baseline Approach Implementation 31](#_Toc5495570)

[4.3.1 Implementation Details 31](#_Toc5495571)

[4.3.2 The Architectures: CNN-2 & CNN-IMU 32](#_Toc5495572)

[4.3.1 Data Loading 35](#_Toc5495573)

[4.3.2 Training 36](#_Toc5495574)

[4.3.1 CNN-IMU Implementation 38](#_Toc5495575)

[4.3.3 Errors and Testing 38](#_Toc5495576)

[4.4 Implementation of Experiments 38](#_Toc5495577)

[4.4.1 Machine Vision Approach 1 38](#_Toc5495578)

[4.4.2 Machine Vision Approach 2 39](#_Toc5495579)

[4.4.3 Machine Vision Approach 3 40](#_Toc5495580)

[Chapter 5 - Results and Discussion 41](#_Toc5495581)

[5.1 Baseline Approach 41](#_Toc5495582)

[5.2 Machine Vision Approach 44](#_Toc5495583)

[5.3 Discussion 48](#_Toc5495584)

[Chapter 6 – Ethics 50](#_Toc5495585)

[Chapter 7 – Limitations, Conclusions and Further ResearchReferences 51](#_Toc5495586)

[Appendix 1 54](#_Toc5495587)

[Glossary 55](#_Toc5495588)

Table of Figures

# Chapter 1 - Introduction

## 1.1 Human Activity Recognition

Human activity recognition (HAR) is the term coined to define an ability to interpret a gesture or motion of the human body, and from this make a determination of the human activity or action being performed [1]. Recognizing a human’s activity, automatically, has become a significant problem in ubiquitous computing, human-computer interaction and human behavioural analysis [2]. Three scientific research contexts HAR finds itself an important component of are: surveillance, healthcare and human computer interaction [1].

### 1.1.1 Applications of HAR

**Surveillance**

HAR has been adopted in surveillance systems at public places, i.e. airports, banks, etc [1]. The findings in [3] confirm that proposed approaches are able to recognize ongoing human-human interactions at the earlier stage [1]. Furthermore, Legion: AR, a system proposed by [4], supplements existing recognition systems with on-demand, real-time activity identification to produce robust, deployable activity recognition [1].

**Healthcare**

The literature indicates that HAR has so far been employed in healthcare systems introduced in hospitals, rehabilitation centres, and even in residential environments [1]. HAR has extraordinary potential within the healthcare sector with a wide range of applications already seen today and is of particular use for aiding the elderly and vulnerable of society. By monitoring the activities of elderly people cared for in rehabilitation centres, HAR can be an effective way of monitoring chronic diseases as well as aiding disease prevention [1]. HAR is also utilised for monitoring patients at home to measure aspects of daily living such as energy expenditure to assist obesity prevention, treatment [5] and lifelogging [1]. Another use case of HAR is its application in monitoring stereotypical motion conditions suffered by children with Autism Spectrum Disorder (ASD) [6]. HAR can also be used for monitoring other behaviours such as those derived from abnormal conditions in cardiac patients [7], and the detection of early signs of illness [8], providing clinicians with an alarm mechanism for early intervention. More healthcare related HAR like fall detection is demonstrated by [9].

**Human Computer Interaction**

The human computer interaction field has seen the most well-known introductions of HAR, being adopted by gaming and exergaming such as Kinect and the Nintendo Wii [1]. The gestures and movements recognized through HAR are used by the machine to carry out specific tasks. HAR is also used in full-body motion-based games for older adults with neurological injury [1].

## 1.2 Types of HAR

There are two main types of HAR: video-based and sensor-based HAR [10]. More specifically, the sensing technologies used in HAR can be classified into the following three categories: RGB camera-based, depth sensor-based and wearable sensor-based [1]. These sensing technologies are discussed further in chapter two. The following figure illustrates the general structure of a HAR system for any of these sensing technologies:

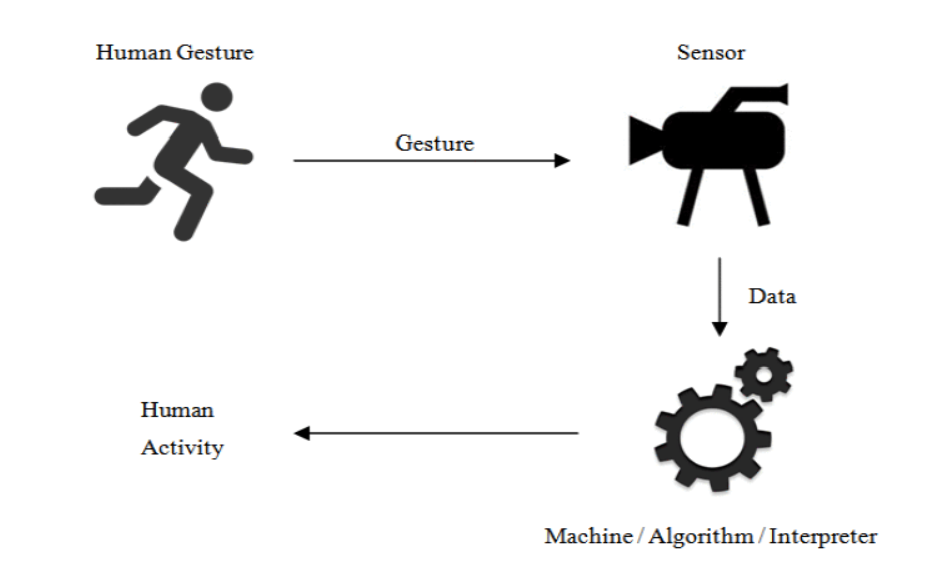


Figure 1: Structure of HAR system

Due to the rapid advancements in ubiquitous computing and the growth in concern for privacy protection, sensor-based HAR via wearable devices has seen itself rise in popularity. Because of this, and the reasons documented in chapter two, this project focuses on sensor-based HAR using wearable devices.

## 1.3 HAR: Machine Learning vs Deep Learning

HAR has most commonly been treated as a pattern recognition (PR) task, and to much avail. Machine learning algorithms such as the support vector machine have produced very satisfying results, most notably in controlled environments and where the volume of labelled data is low [10]. These machine learning methods do, however, rely on heuristic hand-crafted feature extraction which are limited to human domain knowledge. Hand-crafted features are hard to compute and difficult to scale [11], and therefore only shallow features can be learned using this type of approach [10]. In relation to HAR, training an accurate classifier is a particularly difficult task due to the high inter- and intra-class variability of human actions [11]. In other words, humans often perform different tasks differently to each other [11], which makes classification more difficult. Furthermore, using shallow, hand-crafted features reduces the performance of unsupervised learning methods [10]. Therefore, it is clear there are limited classification accuracies and model generalization available to the conventional pattern recognition approaches associated with the HAR problem [10].

The recent surge of pervasive computing has brought with it a rapid development and advancement of deep learning, achieving real success in a variety of domains [12]. In contrast to the conventional PR methods, deep learning is capable of learning high level features which are far more meaningful than the shallow features designed manually in conventional PR methods [10]. Convolutional neural networks (CNNs) have become very popular for their high accuracy in image classification [12]. By training a CNN, feature extraction and classification are combined into an encapsulated, end-to-end approach [11]. With respect to wearable-sensor based HAR, the features extracted through non-linear transformations are learned directly from the raw data, being more discriminative between the different human activity classes than those crafted in PR methods [11]. A drawback to deep learning, however, is its necessity for large amounts of labelled data to train a deep network. Although there are now many publicly available datasets particularly in the domain of object recognition, classification, detection and captioning such as ImageNet and MS-COCO, alternative tasks with a more specific scope may struggle to find labelled datasets on which to train their networks.

This report is structured as follows. Chapter two discusses the three different sensing technologies used in HAR detailing the advantages and disadvantages to each. Chapter two also covers some of the most common approaches to the HAR problem in the domain of time-series data generated from wearable sensors, and depicts the challenges faced by this domain of HAR and the suitability of CNNs for time-series data. Chapter two finally defines the problem being addressed by this project, which experiments with different adaptations to time-series data input to a suitable CNN architecture. Chapter three details the dataset used and describes the baseline approach taken by this project to the HAR problem. It then also details the design of the experiments, which attempt to improve upon the results obtained by the baseline approach. Chapter four is dedicated to the implementation details of this project, the results of which are discussed in chapter five. Chapter six is a piece about the ethics which are adhered to by this project. Chapter seven then draws some conclusions on the work documented by this project and discusses possible future work.

# Chapter 2 - Technical Background & Literature Review

## 2.1 Sensing Technologies

In order to choose a suitable sensing technology on which to design a HAR system, it is useful to discuss each of them and their pros and cons. The following three sensing technologies are discussed below: RGB video, depth sensors, wearable sensors.

### 2.1.1 RGB Video

Video-based HAR accepts videos/images capturing human motion as the input. Where the

camera is attached to the environment, the HAR system processes the image sequences using supervised learning. This involves the previous trainings of a system prior to its application [1]. This training stage consists of feeding image sequences along with the names of the human activity performed in those images into the system [1]. Feature extraction and classification are central components to the conventional HAR systems seen using this sensing technology [1] [\*13,48\*]. The RGB camera has been less preferred to other sensing technologies in the research of HAR [1]. There are a few reasons for this. Firstly, its ability to capture human motion in 3D space is limited [1] [\*39\*]. The performance of a real time HAR system may also suffer due to the high computational cost in the machine processing required to extract human movements from an image [1] [\*28\*] [\*5\*]. The biggest concern, however, regarding the employment of RGB camera data by a HAR system is the issue of privacy, i.e. users of such a system may not be comfortable with consenting to being recorded at all times [1].

### 2.1.2 Depth Sensor

A depth sensor can be referred to as an infrared sensor [1] [\*49\*]. This sends an infrared beam into a scene and recaptures it using the infrared sensor. The depth of an object can be calculated using the distance travelled by the beam [1]. Microsoft’s Kinect sensor is commonly employed as a depth sensor in HAR, according to reviews in [1] [\*33\*]. The Microsoft Kinect sensor is capable of detecting twenty human body joints using a real-world coordinate system [1] [\*40\*], which makes it very suitable for use in the classification of human movements. The literature is adamant that depth sensors are an improvement upon the RGB camera as a sensing technology [1]. The low cost of depth sensors has contributed to a rise in its popularity as a sensing technology for HAR [1] [\*19,20,27,37,39\*]. So too has its high sampling rate and capability of combining visual and depth information [1] [\*10\*]. Furthermore, the recognition processes appear to be far less expensive in comparison to those derived under RGB camera data [1] [\*10\*]. This said, depth sensors have so far been unable to solve the limitation of sensor viewpoint [1] [\*30\*] and sensor obstruction faced by the RGB camera [1][\*39\*].

### 2.1.3 Wearable Sensor

HAR via the use of wearable sensors requires only the subject wearing single or multiple wearables on their body [1]. The sensors typically found in these wearables are 3-dimensional accelerometer, gyroscope and magnetometers as well as temperature sensors. The emergence of wearables from smartphones (most smartphones now come with an accelerometer, gyroscope and magnetometer) to fitbits has opened many avenues for HAR. Wearables can also overcome the barriers of limited sensor viewpoint and sensor obstruction already discussed for RGB cameras and depth sensors. Additionally, wearables have the potential to act as a non-obtrusive [1] [\*26\*] solution to HAR as they have already proven to be effective in monitoring other aspects of daily living without infringing on the user’s quality of daily living. Further advantages of wearable sensors are their low monetary cost and high power efficiency [1] [\*46\*]. There are still some challenges facing wearable sensors, however. The accuracy of activity recognition using a wearable sensor may not be sufficient as current wearable sensor-based HAR systems require subjects to wear multiple sensors across different body parts [11] [1][\*23\*]. This is undoubtedly a sub-optimal solution to HAR as it is quite inconvenient and mildly intrusive for a subject to be required to wear multiple sensors across different body parts.

## 2.3 Sensor Modalities

Most HAR approaches work with just one specific type of sensor only. [10](Chavarriaga et al., 2013) suggests that we classify the sensor modalities into the following categories: *body-worn sensors, object sensors, ambient sensors.* These sensor modalities are further detailed below:

**Body-worn Sensor**

The body-worn sensor is of the most commonly used sensor modalities in HAR. As their name suggests, these sensors are most often worn by the user and include accelerometers, gyroscopes and magnetometers. These sensors are widely distributed throughout ubiquitous computing and are now very often found in watches, smart phones, bands, helmets and glasses. Body-worn sensors are very frequently used for deep learning based HAR [2] [11] [13] [14]. This related work shows that body-worn sensors are primarily adopted for recognizing activities of daily living (ADL) and sport. One notable point regarding the use of body-worn sensors in deep learning based HAR is that it is the original signal produced by the sensors that is used as the input to the network, as opposed to the approach of traditional machine learning methods where the inputs are statistical/frequency features extracted from the motion data.

**Object Sensor**

Object sensors are different to body-worn sensors in that they infer human activity by detecting the movement of a certain object [10](Chavarriagaetal.,2013). Therefore, object sensors are placed on specific objects to detect their movement. For example, the activity of drinking water can be detected by placing an accelerometer on a cup, [10]. Radio frequency identifier (RFID) tags are often used and mainly employed in smart homes [10] (Vepakomma et al., 2015; Yang et al., 2015; Fang and Hu, 2014) and medical facilities [10] (Li et al., 2016b; Wang et al., 2016a). RFID tags are used as they can produce more fine-grained information aiding in the recognition of more complex activities [10]. Object sensors are not as popular as body-worn sensors for HAR as they are difficult to deploy in real-world environments, however, there is an emergence of using object sensors combined with other types for the recognition of more complex activities [10] (Yang, 2009).

**Ambient Sensor**

In contrast to object sensors which measure an object’s movement, ambient sensors attempt to capture changes in the environments. Human activity is inferred by capturing the interaction between humans and the environment. Radar, pressure sensors, sound sensors, and temperature sensors are all different examples of ambient sensors. The work with ambient sensors indicates they are usually embedded in users’ smart environment (smart home) and are used to recognize activities of daily living and hand gestures [10] (Lane et al., 2015; Wang et al., 2016a; Kim and Toomajian, 2016). Like object sensors, the deployment of ambient sensors is not easy and so they are often less preferred to other sensor types. Another drawback to ambient sensors is that they are very easily affected by the environment, and therefore, only specific types of activities are accurately inferable by ambient sensors.

**Hybrid Sensor**

The combination of sensor types has been shown to potentially improve the accuracy in HAR. Ambient sensors used alongside object sensors are advantageous as they provide information on object movements as well as the state of the environment. In [10] (Vepakomma et al., 2015), a smart home environment is designed in which a wide range of complex and fine-grained activities of multiple occupants are recognizable via the use of body-worn, object and ambient sensors. It is clear that using multiple sensor modalities can yield superior information of human activities. The work also shows that it is a deployable solution to HAR in certain environments like smart homes.

## 2.4 CNNs and HAR

Convolutional Neural Networks have been used extensively in the field of computer vision for their effective feature extraction and pattern recognition abilities. They have been widely adopted by deep learning and have produced very promising results in image classification, speech recognition and text analysis [10]. CNNs are of particular use in image classification as convolution leverages three important ideas: sparse interactions, parameter sharing and equivariant representations [15] [\*23\*], [datadriveninvestor?]. With these properties, CNNs can extract smaller features of greater significance, whilst at the same time reducing the storage requirements compared to those more densely connected neural networks [15]. In addition to this, the depth and breadth of convolutions are adjustable, making it easier to train CNNs compared to alternative feedforward neural networks [15]. CNNs do, however, raise the concern of overfitting the training set, particularly in image recognition. If overfitting has occurred, it means that the CNN is also learning the obsolete background features of the training images, aiding the classification accuracy of the training set [15]. This is not desirable as it means that the trained model will not generalise well, i.e. it has failed to capture the more general characteristics of the training set and so, on unseen data, the model will not produce a classification accuracy close to that found on the training set.

When used for the classification of time-series data, like in HAR, CNNs have two advantages over other models: local dependency and scale invariance [10], [14]. Local dependency refers to the correlation between nearby signals [10]. Scales invariance means the output is not sensitive to a variance in paces or frequencies in the input. When using CNNs as a solution to the HAR problem, the concepts of *input adaptation, pooling* and *weight sharing* must be considered [10].

### 2.4.1 Input Adaptation for CNNs

The sensors used in HAR, obviously, do not produce image data. These sensors produce time-series readings. The most commonly gathered data in wearable sensor-based HAR are accelerometer, gyroscope and magnetometer time-series signals. More specifically, these are 3-axial (x, y and z direction) 1D readings along the temporal dimension [10]. These readings must be adapted first before a CNN can be applied to them. The inputs must first be adapted to form a *virtual image* [10]. There are two ways of achieving this: *data-driven* and *model-driven*.

* **Data-driven:** This approach is straightforward and easy to implement. Each dimension is treated as a channel on which a 1D convolution is performed, i.e. each 1D sensor is treated as a 1D image [10]. Although uncomplicated, this approach does have a drawback: it does not account for dependencies between dimensions and sensors, which may impact performance [10].
* **Model-driven:** This approach looks to apply a 2D convolution, and so resizes the inputs into a 2D virtual image [10]. The advantage of this approach is that it can incorporate the temporal dependencies between sensors [10]. This conversion from time-series to image data is, however, a non-trivial task and requires domain knowledge [10].

### 2.4.2 Pooling

Pooling is often paired with convolution in CNNs. The most common types of pooling are max or average pooling. Convolution is so often combined with pooling operations to avoid overfitting [10]. Each time an input image is pooled, the image size is reduced (usually to half the size). Therefore, pooling helps to reduce the training time on large training sets as it reduces the number of parameters to be tuned [10] (bengio, 2013).

### 2.4.3 Weight Sharing

## 2.6 Common Approaches to HAR

(Few take a machine vision approach – only found one paper)

Many different approaches have been taken to the HAR problem and many works tackle HAR from different angles. [16] explores, compares and contrasts the state-of-the-art deep learning methods for HAR using wearable sensors regarding performance. [16] gives great insights into the suitability of each model with respect to specific tasks in HAR. The three main deep learning architectures assessed in [16] are Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs) and regular Deep Feed-Forward Neural Networks (DNNs). It is observed from [16] RNNs that outperform CNNs significantly on activities that are short in duration but have a natural ordering. It is clear that the performance of a recurrent approach benefits largely from its ability to contextualise observations across long periods of time. It is, however, recommended that CNNs are used for more prolonged and repetitive activities such as walking/running. Although some RNNs were found to perform similarly to or even better than CNNs in this environment, the *average* performance of CNNs in this domain suggest a higher likelihood that the practitioner finds a suitable configuration using a CNN architecture. Furthermore, through the experiments in [16] it is recommended to explore different learning rates prior to optimizing the network architecture. This is due to the experiments in [16] demonstrating that a change in learning parameters has the largest impact on performance. The work with DNNs in [16] indicate that they require a significant investment in the exploration of parameters, showing a notable spread between peak and median performance. It is concluded from [16] that the use of CNNs or RNNs are a more sophisticated approach to DNNs, showing a smaller spread in performance and a higher likelihood of finding a configuration that works sufficiently, particularly in the case of CNNs.

[2] focuses on the HAR problem where the inputs are multi-channel time-series measurements gathered from inertial measurement units worn on different parts of the body, and the outputs are predefined human activities, e.g. walking, running, sitting etc. The architecture used by [2] is a deep CNN, automating feature learning from the raw inputs. As so often used in HAR, labelling the inputs via supervised learning produces learned features with more discriminative power [2]. [2] uses benchmark datasets like the Opportunity Activity Recognition dataset and Hand Gesture dataset to verify that the unique advantages to CNNs propel their CNN architecture to outperform other state-of-the-art machine learning solutions to HAR such as SVM, KNN, Means and Variance, and Deep Belief Network.

Similar to [2], [11] focuses on the HAR problem where the inputs are multi-channel time-series measurements. A novel CNN is proposed by [11] in which the network used processes the sequence measurements from different body-worn devices separately. This follows the idea of a wider rather than a deeper network, where the architecture consists of multiple branches, each processing the data from one specific inertial device only. This is done to provide some robustness against the inertial devices being asynchronous or having slightly different characteristics. With each branch providing an IMU-specific, intermediate representation of the data, a global representation is found via fully connected layers. 1D temporal convolutions and pooling operations are performed on the input’s sequences. [11] evaluates the proposed architecture on three benchmark datasets including the Opportunity Activity Recognition and Pamap2 datasets, outperforming the state-of-the-art. [11] advises that the capabilities of CNNs are improved by applying convolutions per sensor and per body worn device.

[17] proposes the use of 2D kernels in both convolutional and pooling layers to capture spatial dependency over sensors in addition to local dependency over time (already provided by the temporal convolution). [17] demonstrates a high performance of this multi-modal CNN, compared to state-of-the-art methods by experimenting on benchmark datasets.

As in [17], [18] applies 2D convolution and pooling to capture both spatial and local dependency. However, it is not the raw time-series data which is passed into the network. [18] carries out pre-processing steps to adapt the time-series input sequences into a 2D virtual image. It first uses an algorithm to create an image of sensor channels where each sensor channel is placed either above or below all other sensor channels at least once. This is done to allow the deep CNN to extract hidden correlations between neighbouring signals. This generates a new signal image. The DFT of each channel in this signal image is found to generate a new activity image. The activity image is the input which is passed to the architecture. According to [18], this approach outperforms the state-of-the-art in terms of recognition accuracy and computational cost.

[19] presents CNNs for multi-modal data (multi-sensor, e.g. accelerometers and gyroscopes) where it introduces both partial weight sharing and full weight sharing to the CNN models in a manner that both modality-specific characteristics and common characteristics across modalities are learned from multi-modal data. The results in [19] indicate that weight sharing can improve the performance of a CNN.

[15] takes an alternative approach to HAR with the use of photoplethysmography sensor data gathered from wearable devices. [15] looks to simplify the wearable approach to HAR by experimenting with wrist-mounted optical sensing (used usually for heart rate determination) to see if it can provide data useful for activity recognition. In contrast to many of the papers discussed, [15] takes a machine vision approach to the HAR problem, using the plots of the optical signals to produce activity classifications. More specifically, [15] implements transfer learning to retrain the penultimate layer of a pretrained CNN, using time-series *images* of the photoplethysmography signals as the inputs to the network. [15] achieves an average accuracy of 75.8%, which is a competitive result, suggesting that the overall design of activity monitoring and classification systems could be simplified to wearables based on optical measurements only. However, implementing a design of optical sensor only in an activity classification system does lead to a trade off in classification performance.

## 2.7 Project Outline

As can be seen, there are many different approaches to the HAR problem, with many variations in architectures and how the data is represented. After a careful examination of the literature on HAR, it can be concluded that CNNs are the most effective network used in HAR to date. After an in-depth study of the literature surrounding HAR, this project takes the following form.

Firstly, this project attempts to reimplement the architecture in [11] to establish a baseline approach to HAR. Establishing a baseline approach is an important step for this project for a number of reasons. With the undertaking of two novel topics in deep learning and HAR, it is important to find a strong grounding in these areas. With the help of the baseline paper, [11], a good baseline approach to HAR using deep learning can be achieved. This can then act as a platform on which to experiment and it can also serve as a benchmark to compare new approaches with.

A benchmark dataset, Pamap2 [20] accessed at [21], as is used in [11] is used to implement and test this baseline approach. Once a working network is established and a model is successfully trained, where the inputs are multi-channel time-series measurements from inertial sensors, this project then experiments with a machine vision approach to HAR. The concept executed by [15] is explored, where plots of sensor signals are saved as images and are then passed as inputs to the network. In [15], just one optical signal is plotted, providing an average classification accuracy of 75.8%. This project applies a similar approach to inertial measurement data where multiple triaxial sensors are used. In theory, the approach taken by [15] should be enhanced where multiple signals are plotted, providing more information than just a single optical signal as in [15]. This project explores and compares different ways of plotting the data and how best to combine the data of multiple inertial sensors. It also experiments with different learning parameters in order to determine what set up is needed to best support this machine vision approach. One final note: the approach in [11] is chosen as the baseline approach for this project as the use of separate branches may provide flexibility when experimenting with different ways of passing the plots of multiple inertial devices to the network.

## 2.7 Modality Transformation

Modality transformation refers to the conversion of data from a source mode to a target mode [12].

# Chapter 3 - Design of Solutions

In this chapter, 3.1 first describes the dataset used by this project and the reasons why it is suitable. The baseline approach that this project attempts to implement is then covered in 3.2. This section outlines some key technical concepts applied by the benchmark paper, [11], to their approach before describing the actual architecture used. 3.3 then addresses exactly how this project will build on the baseline approach implemented, experimenting with a machine vision approach to HAR.

## 3.1 The Dataset – Pamap2 Dataset

The dataset used by this project is freely available for academic research and has often been used as a benchmark dataset in HAR research [11], [1]. Of the public datasets available, the Pamap2 dataset has one of the highest sampling frequencies (100 Hz), along with one of the highest number of recorded measurement samples, 2,844,868 [1]. It is also one of the few datasets with the luxury of having an accelerometer, gyroscope and magnetometer all in use, along with temperature sensors and a heart rate monitor.

### 3.1.1 Hardware Setup

For the measuring of the different activities of daily living performed in this dataset, three Colibri wireless inertial measurement units (IMUs) were used. The position of these IMUs on the body of each subject during the data collection was as follows:

* 1 IMU over the wrist of the dominant arm
* 1 IMU on the chest
* 1 IMU on the ankle of the dominant leg

The sampling frequency of the IMUs is 100 Hz.

In addition to the three IMUs, each subject also wore a heart-rate monitor, BM-CS5SR from BM innovations GmbH. The sampling rate of the heart rate monitor used is ~9 Hz.

Finally, a companion unit was used: Viliv S5 UMPC. This companion unit has an Intel Atom Z520 CPU (1.33 GHz) and 1 GB of RAM. The labelling of the different activities performed during the data collection was done via a GUI running on the Viliv.

### 3.1.2 Subject Information

A total of nine subjects participated in the collection of this dataset. Eight of the participants were male with just one female participant, and the age spread of the subjects at the time of the data collection was 27.22 ± 3.31 years. The subjects had a BMI of 25.11 ± 2.62 . The table below shows the relevant information of each subject:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Subject ID | Sex | Age (years) | Height (cm) | Weight (kg) | Resting HR (bpm) | Max HR (bpm) | Dominant Hand |
| 101 | male | 27 | 182 | 83 | 75 | 193 | right |
| 102 | female | 25 | 169 | 78 | 74 | 195 | right |
| 103 | male | 31 | 187 | 92 | 68 | 189 | right |
| 104 | male | 24 | 194 | 95 | 58 | 196 | right |
| 105 | male | 26 | 180 | 73 | 70 | 194 | right |
| 106 | male | 26 | 183 | 69 | 60 | 194 | right |
| 107 | male | 23 | 173 | 86 | 60 | 197 | right |
| 108 | male | 32 | 179 | 87 | 66 | 188 | left |
| 109 | male | 31 | 168 | 65 | 54 | 189 | right |

### 3.1.3 Data Collection Activity Protocol

Each subject followed a protocol containing twelve different activities of daily living. In addition to this, a list of optional, additional activities was given to the subjects. From the list of optional activities, a total of 6 different activities were performed by some of the subjects in addition to the protocol. The list of optional activities contained a wider range of activities from household and everyday activities to sport activities. Below shows a list of the activities along with their label:

|  |  |
| --- | --- |
| Activity ID: | Activity |
| 1 | Lying |
| 2 | Sitting |
| 3 | Standing |
| 4 | Walking |
| 5 | Running |
| 6 | Cycling |
| 7 | Nordic walking |
| 9 | Watching TV |
| 10 | Computer work |
| 11 | Car driving |
| 12 | Ascending stairs |
| 13 | Descending stairs |
| 16 | Vacuum cleaning |
| 17 | Ironing |
| 18 | Folding laundry |
| 19 | House cleaning |
| 20 | Playing soccer |
| 24 | Rope jumping |
| 0 | Other (transient activities) |

The dataset documentation instructs that any data labelled with an ID of 0 should be ignored in any analysis. This data is that which was collected when the subject was transitioning from one activity to another, and so is not indicative of any specific activity.

### 3.1.4 The Collected Data

In the actual collection of the data, some data was lost. This is due to two main causes: data dropping and problems with the hardware setup. Data dropping may have occurred from using wireless sensors. It is also important to note that the “real” sampling frequency of the IMUs was 99.63 Hz, 99.89 Hz and 99.65 Hz for the hand, chest and ankle IMU respectively.

In total, over ten hours of data was collected, from which almost eight hours is representative and labelled as one of the 18 activities performed during the data collection. The below histogram shows the distribution of activities across all subjects with respect to time in seconds:

### 3.1.5 Data Format

The synchronized and labelled raw data from all sensors (3 IMUs and the HR-monitor) is merged together into one single file per subject and per session (protocol/optional). These data (.dat) files are made available at [21]. These data files each have 54 columns per row. The data under each column is summarised below:

* 1 timestamp
* 2 activity ID (label)
* 3 heart rate (bpm)
* 4-20 IMU hand
* 21-37 IMU chest
* 38-54 IMU ankle

The data under each column of the IMU sensory data is summarised below:

* 1 temperature (°C)
* 2-4 3D-acceleration data (), scale: ±16g, resolution: 13-bit
* 5-7 3D-acceleration data (), scale: ±6g, resolution: 13-bit
* 8-10 3D-gyroscope data (rad/s)
* 11-13 3D-magnetometer data (µT)
* 14-17 orientation (invalid)

The dataset documentation informs that the orientation data collected is invalid. All missing sensory data due to wireless data dropping are replaced with NaN in the data files. Because the HR-monitor has a much lower sampling frequency of 9 Hz compared to the sampling frequency of the IMUs (100 Hz), there are far less HR data values recorded. The missing HR values are also filled in with NaN in the data files.

Finally, the Pamap2 dataset documentation also advises that the second accelerometer is not calibrated exactly with the first one. Due to the high impacts of more strenuous activities like running where an acceleration exceeds 6g, the second accelerometer of scale ±6g sometimes gets saturated. Therefore, it is advised that the data of the first accelerometer with a scale of ±16g is used.

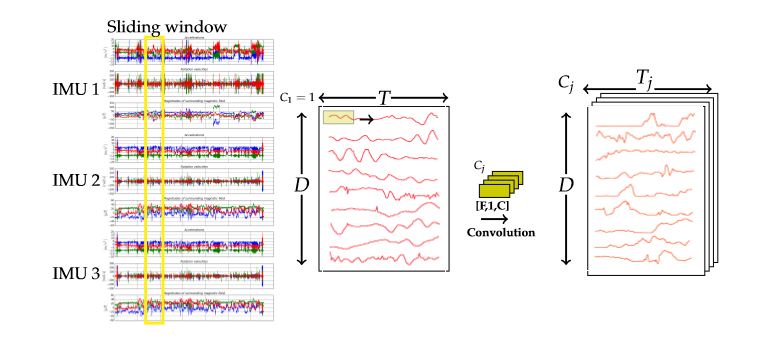
## 3.2 Baseline Approach

As mentioned, this project first attempts to replicate the architecture designed by [11]. Before describing the novel architecture designed by [11], there are a few key concepts to address. This architecture uses a CNN. As discussed in chapter 2.4, a CNN is a hierarchical structure which combines convolutional operations that use learnable filters and non-linear activation functions with downsampling (pooling) operations and classifiers [11]. Different features are extracted at different layers of a CNN. CNNs are capable of extracting more abstract features by stacking convolutional layers and downsampling their outputs [11]. This way, CNNs can also remain invariant to distortions [11].

In this approach, the input to the CNN is a stack of the sequences produced by the sensors. More specifically, a single input to the CNN is a segment of these sequences where the length of the segment is some fixed temporal duration. Therefore, the input is a 2D matrix where each row corresponds to a single 1D sensor signal and each column corresponds to a single sample of data for each sensor. In summary, the input is a 2D matrix of *T* measurements for each of the *D* sensors. Following [11], and as commonly used with time-series data, a *sliding window approach* is used to generate these input sequence segments.

### 3.2.1 Sliding Window

With a stacked sequence of *d* = 1, 2, …, D sensors, a sliding window of size *T* is moved forward over the sequences along the temporal axis with some frame-shift, *s*. Each time the window is moved forward by *s*, the sequence segment enclosed by the window is captured and acts as an input to the network [11]. These segmented sequence inputs are therefore of size [*T, D*] [11]. By using a small value of s, multiple windows for even just a single activity can be extracted [11]. Furthermore, generating a large number of inputs is important for training a CNN effectively [11]. The CNN’s inputs extracted using a sliding window approach is illustrated below:



Caption: a *virtual image*  of the sensory data is generated using the sliding window technique.

(As can be seen, a *virtual image* of the sensory data has now been generated using the sliding window technique.)\*can remove

The baseline approach of this project sees the sliding window technique applied in two ways.

**Sliding Window Method 1**

The first operates exactly as just explained above, where the window is placed over all of the sensor channels available, creating one *virtual image* input which represents the sensory data of all IMUs together. Where *D* is the entire number of sensors available across all IMUs and *T* is the window size, the input image generated is of size [*T, D*].

**Sliding Window Method 2**

The second way this method is applied by the baseline approach is creating a *virtual image* *Dinput* which represents the sensory data of a single IMU only. Where is the number of sensors available in IMU i and *T* is the window size, the input image generated for IMU i is of size [*T,* ]. Therefore, in the case of the Pamap2 dataset where there are *m* = 4 IMUs available (3 IMUs and a HR-monitor), the sliding window technique must produce *m* = 4 input images each time the window is moved by a step of *s*. This can be thought of as *m* = 4 sliding windows, each moving at the same rate over a specific IMU.

The following sections describe how these inputs are used in a CNN.

### 3.2.2 Temporal Convolution and Pooling

The architecture used by this approach largely consists of temporal and pooling operations.

The *virtual image* inputs can also be expressed as feature-map inputs to the convolutional layers. In CNNs, each convolutional layer convolves its feature-map inputs with *C* filters along the temporal axis [11]. Consider layer *i*. A single input to layer *i* is a feature-map of size [*T, D, C*] [11]. Now consider layer *j*, connected to layer *i* via a set of filters of size [*F, 1,* ] and biases [11]. A temporal convolution of each sensor d (1D) can be expressed by the following equation [11]:

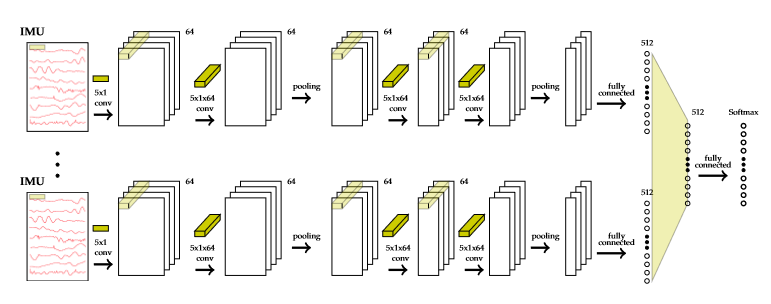
*σ* is the activation function. As can be deduced from the above equation, the filters are shared among all D sensors.

Pooling operations reduce the size of the feature-map with respect to the temporal axis. The most common pooling operation, max-pooling, is used by [11] to induce a temporal robustness. For a single channel *c*, a max-pooling operation between two layers, *i* and *j*, simply finds the maximum value among a set of p values. Where is the feature map input to pooling layer *i*, the max pooling operation can be expressed by the following equation:

### 3.2.2 The Architecture

**CNN-IMU Network**

The novel architecture proposed by [11] processes the time-series data of multiple IMUs separately. This is based on a sensor setup, demonstrated by the Pamap2 dataset, where the subject wears multiple IMUs at different locations around their body. [11] refers to this architecture as a CNN-IMU network, and is illustrated below:



As seen in figure xnumx, this architecture consists of parallel branches, one per IMU. Therefore, an input image is generated for each branch (using only the sensor sequences of a specific IMU) and these images are processed by their respective branches simultaneously. This architecture uses the sliding window method number two as described in 3.2.1 to generate the input images for each branch.

Each branch contains *B* blocks, with each block having two stacked [5 x 1] temporal convolutions followed by a subsequent [2 x 1] max pooling operation. It uses temporal convolutional layers to locate temporal-local features in the inputs [11]. Fully-connected layers are implemented to connect the local features and generate a global representation of the data [11]. Each branch finishes with a fully connected layer [11]. These layers are processed simultaneously for each IMU to increase the descriptiveness of the network [11]. The use of separate branches for each IMU is also implemented to increase the robustness of the network to slightly asynchronous IMUs or IMUs having varying characteristics [11]. The network then combines the intermediate representations of each branch into a global representation of the data using a subsequent fully connected layer [11]. As this is a classification task where an input sequence segment can represent only one activity, a softmax activation is used to generate probabilities for each class. [11] uses the cross-entropy loss between the estimated probabilities and the target label for training the model. Finally, dropout is also applied to all fully connected layers, apart from the classification layer. The number of neurons used per layer is *C* = 64 [11].

This project attempts to implement the CNN-IMU architecture just described.

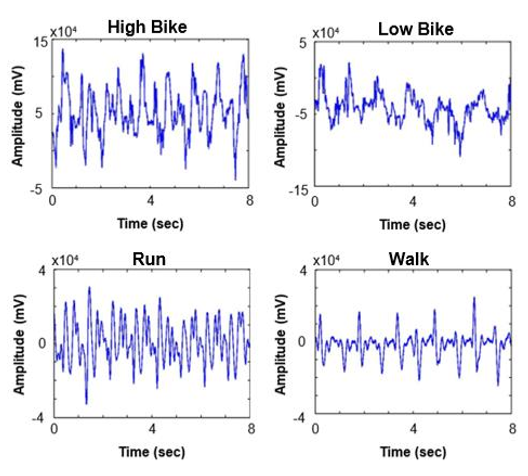
**CNN Baseline Network**

[11] also evaluates a slightly simpler CNN baseline network, which takes in a *single* input sequence of the sensory signals from *all* of the IMUs, not just a single IMU. Therefore, this network uses sliding window method 1 as described in 3.2.1 to generate the single input image for this network. This architecture is the type which is more commonly seen in HAR [2], and can be thought of as the CNN-IMU architecture described above with just a single branch to process the data of all IMUs. Thus, this CNN baseline network consists of *B* blocks of two [5 x 1] temporal convolutions followed by a [2 x 1] max pooling operation, with three fully connected layers at the end.

This project replicates both architectures, CNN-IMU and CNN baseline, proposed in [11] as the baseline approach to this project. This project then experiments with modality transformation of the sensory data inputs, from a *virtual* 2D image of time-series measurements into an actual image. The design of these experiments is described below.

## 3.3 Experiment Design

This project carries out three experiments where, following [15], a machine vision approach is taken. [15] designs its own data collection where photoplethysmography (PPG) recordings are taken from 8 subjects (3 male, 5 female), with a mean age of 26.5 years [15]. The PPG recordings are taken from the subjects during controlled exercises on a treadmill and exercise bike using a wrist-worn PPG sensor [15]. The four exercises performed in this data collection are: walking, running, low resistance cycling and high resistance cycling [15]. The signals gathered are then segmented into smaller 8-second (used in [\*18\*]) long time-series windows. Similar to the baseline approach of this project, a sliding window function is used to step through the data with a frame shift, *s*, of 2 seconds, each time saving a new plot of the time-series PPG data plotted for 8 seconds. To clarify, it is a machine vision approach being taken here. Therefore, the input data created by [15] are actual images (saved as .jpg files) and not time-series vectors. [15] gives the following example for the plots of the PPG data for each of the four activities:



[15] uses the concept of transfer learning to retrain the penultimate layer of a pretrained CNN (a trained network for image classification). However, [15] advises that this approach may be a limitation to the potential classification accuracy of the system, suggesting that building a complete neural network from scratch could yield better results.

This project attempts to apply this same machine vision approach but to the time series data gathered from multiple IMUs. Following the suggestion made by [15], this project then uses the input data to train a network from scratch, as opposed to using transfer learning. The CNN baseline and CNN-IMU architectures designed for the baseline approach of this project are also used to implement the experiments detailed in this section. This project experiments with three specific machine vision approaches, detailed below.

### 3.3.1 Machine Vision Approach 1 (called 2 in code)

This approach uses a CNN-IMU architecture, where there are separate branches for each of the IMUs and the HR-monitor. Therefore, there is a total of four branches. Due to there being only a single sensor channel for the HR-monitor, the single channel time-series data is passed in as the input to the first branch (branch for the HR-monitor) just as done in the CNN-IMU baseline approach. Another reason for this is the lower sampling rate of the HR-monitor in comparison to the IMUs. Because the sampling rate is far lower than that of the IMUs, a plot of the HR-monitor data does not provide enough information to justify using plots of the HR-monitor data as inputs to the HR branch, particularly where the window size is chosen to fit the sampling rate of the IMUs.

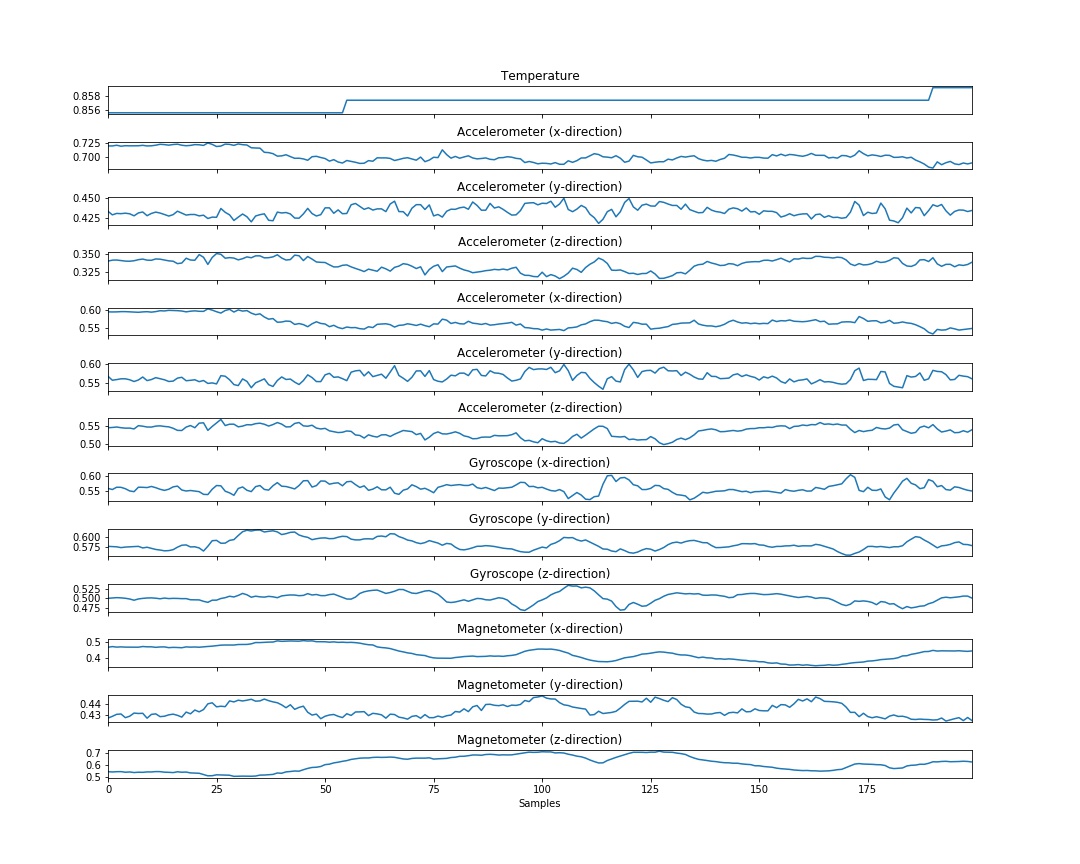
Just as in the baseline approach, a sliding window technique is used to generate the plots. This time, instead of generating a *virtual* image of size [*T, D*], where *T* is the window size and *D* is the number of sensor channels, an actual image is generated consisting of *D* subplots. Each subplot plots the time-series data of sensor channel *d* against time for some number of samples (window size) *T*.

Recall that each IMU consists of the following sensors:

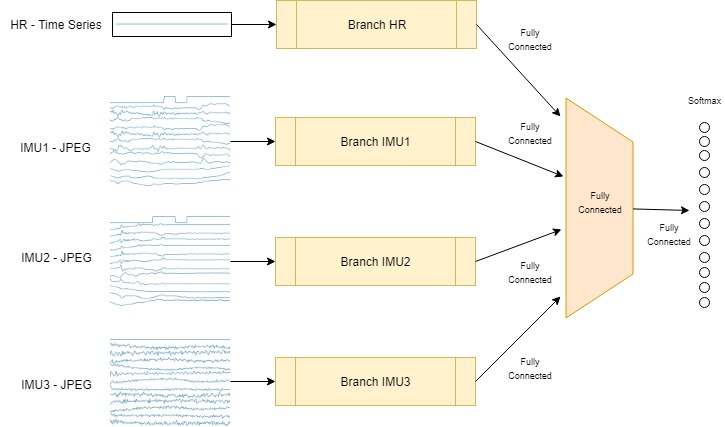
* temperature (°C),
* 3D-accelerometer (), scale: ±16g, resolution: 13-bit,
* 3D-accelerometer (), scale: ±6g, resolution: 13-bit,
* 3D-gyroscope (rad/s),
* 3D-magnetometer (µT).

Although the dataset documentation advised that the accelerometer of scale ±6g gets slightly saturated for the more explosive activities which cause larger accelerations, this data is still used in both the baseline approach and experiments of this project. This is due to the fact that this data is used by [11], which acts as the benchmark for this project.

As can be seen, there are 13 sensor channels per IMU. This approach assigns each sensor signal with its own subplot so that each signal is plotted with its own scale. This way, the finer details in the signal’s fluctuations is preserved. Therefore, each input image has 13 subplots, each with their own scale. This way, the behaviour of each signal with respect to time can be accurately represented. This can be seen more clearly in the below example, where a segment is plotted for T = 200 for one IMU:

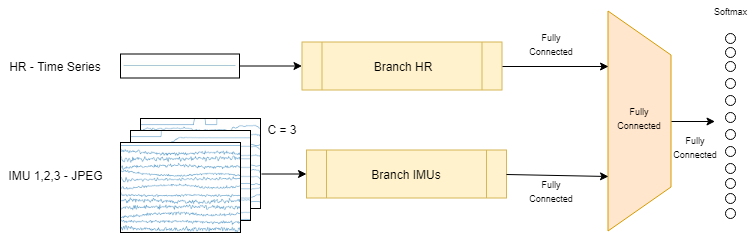


The script which prints and saves these images ensures to remove all non-salient features such as axis labels, grid lines and ticks, legends etc. The image is saved as a 640 x 480 .jpg file. As there are three IMUs in the Pamap2 dataset, three of the above images will be generated per frame shift, *s* (one image per IMU). For clarity, the following image illustrates how the IMU image inputs and the HR time series data are fed to the CNN-IMU network:



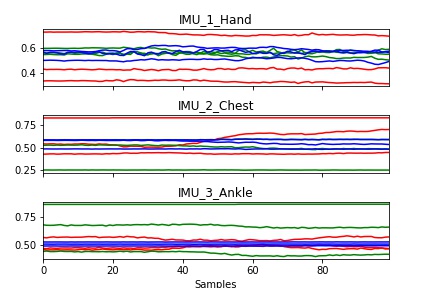
### 3.3.2 Machine Vision Approach 2 (called 1 in code)

This approach is a variation of the first experimental approach. In this case, the JPEG images are added together to form an image with three channels. With three IMUs represented by the one image, this image is then passed to a branch. The HR data is dealt with in the same fashion as the first machine vision approach, as illustrated below:



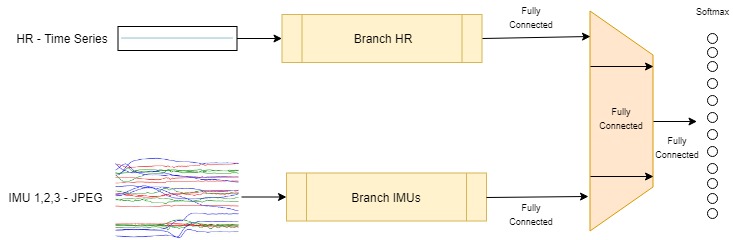
### 3.3.3 Machine Vision Approach 3

This approach is different to the previous two in three ways. Firstly, the sensor channels of all three IMUs are plotted on a single image. Secondly, all sensor channels of one IMU are plotted on a single subplot. Therefore, each image consists of three subplots, one per IMU. Finally, an RGB image is generated where plots of acceleration data are red, plots of gyroscope data are green, and the plots of magnetometer data are blue. The below image demonstrates this setup:



This approach attempts to account for spatial dependencies between sensor signals. This does, however, lead to a trade off in the signal detail in the image. As can be seen in figure fxxxf, there is a loss in the finer detail of the sensor signals with respect to smaller fluctuations due to all sensor signals from a single IMU being plotted using the same y-axis.

Because all IMUs are represented on one image, a single branch is all that is needed for all three IMUs in the CNN-IMU architecture. An additional branch is needed to process the data from the HR-monitor. Just as is done in the previous machine vision approach, the single channel time-series data from the HR-monitor is passed in as the input to a separate branch in the CNN-IMU architecture. This is illustrated below:



# Chapter 4- Implementation and Testing

In the previous chapter, the Pamap2 dataset that is used by this project is described. Chapter three then outlines the baseline approach taken by this project and describes the experimental work in the modality transformation of multi-channel time-series data to image data undertaken by this project. This project uses the PyTorch deep learning framework to implement the designs as described in chapter three. All training of deep learning models is done on a cuda GPU.

This chapter first describes the data pre-processing steps necessary to convert the raw data files provided by the Pamap2 dataset to clean, normalized training, validation and test datasets. 4.3 then describes the implementation of the two baseline architectures: CNN-baseline and CNN-IMU. 4.4, 4.5, and 4.6 each detail the implementation of the three experimental approaches described in 3.3.1, 3.3.2, and 3.3.3 respectively.

## 4.1 Dataset Design Protocol

## 4.2 Data Pre-Processing

The data collected in the Pamap2 dataset comes from nine participants. Recall that the dataset’s protocol contains *K* =12 classes: *lying, sitting, standing, walking, running, cycling, nordic walking, ascending stairs, descending stairs, vacuum cleaning, and rope jumping*. Although there are six more optional activities which were performed by participants 1, 5, 6, 8 and 9, this project focuses on just the *K* = 12 classes mentioned above, following the benchmark paper [11].

The Pamap2 dataset contains .dat files for each subject's readings in a protocol folder, with an optional folder containing .dat files of the additional readings of optional activities from subjects 1, 5, 6, 8, 9. In total, there are 14 .dat files of raw sensor data. As seen in chapter 3.1, each .dat file contains the synchronized and labelled raw data from the sensors of the three IMUs and the HR-monitor. Each data-file contains 54 columns per row. As a reminder, the information in each column is as follows:

* 1 timestamp
* 2 activity ID (label)
* 3 heart rate (bpm)
* 4-20 IMU hand
* 21-37 IMU chest
* 38-54 IMU ankle

The data under each column of the IMU sensory data is summarised below:

* 1 temperature (°C)
* 2-4 3D-acceleration data (), scale: ±16g, resolution: 13-bit
* 5-7 3D-acceleration data (), scale: ±6g, resolution: 13-bit
* 8-10 3D-gyroscope data (rad/s)
* 11-13 3D-magnetometer data (µT)
* 14-17 orientation (invalid)

The dataset declares that the orientation readings are invalid in this data collection. These columns must therefore be removed. The NaN values in the data files (due to wireless data dropping or as a result of the lower sampling frequency of the HR-monitor) must also be replaced with suitable data values. With the orientation data removed, this dataset holds measurements from *D* = 40 sensors from *m* = 4 separate devices (3 IMUs and a HR-monitor).

Following [11], the recorded measurements of participant 5 are used as a validation dataset and the recordings of participant 6 are used as a testing dataset. The rest of the dataset is used for training. In order to generate these training, validation and test datasets which hold only clean and necessary data, the raw data files go through a pre-processing stage. The pre-processing script iterates through the data file of each subject and passes the subject’s data as a 2D matrix to a processing function. This function returns a 1D vector of labels and a 2D matrix of processed sensory data (*D* = 40 columns). The label vector returned is then added to a training/validation/test label vector, depending on which subject the data is from. Similarly, the sensory data matrix returned is added to a training/validation/test data matrix, depending on which subject the data is from. This is done so that at the end of the script, a single sensory data matrix and corresponding label vector exists for all training, validation and testing. The following steps are taken in the processing function to process the 2D raw data matrix passed:

* The timestamp column, labels column and sensory data columns are divided into separate time, label and sensor data matrices.
* All data which represents transient activity (raw data label 0) should not be used in any analysis. Furthermore, the data labelled with an optional activity is not used in the analysis of this project (following [11]), and so this data is also removed.
* There are now *K* = 12 action classes left in the data, but with labels ranging from 1 – 24. The data is therefore relabelled so that the data is labelled from 0 – 11, with each label corresponding to the following activities:

|  |  |
| --- | --- |
| Label | Activity |
| 0 | Rope Jumping |
| 1 | Lying |
| 2 | Sitting |
| 3 | Standing |
| 4 | Walking |
| 5 | Running |
| 6 | Cycling |
| 7 | Nordic Walking |
| 8 | Ascending Stairs |
| 9 | Descending Stairs |
| 10 | Vacuum Cleaning |
| 11 | Ironing |

* Sensory data columns holding orientation data are removed.
* The label matrix is cast to type integer (desirable for when passed to a network).
* The HR NaN values recorded as a result of the lower HR sampling frequency are filled in according to the last non-NaN value recorded.
* The signals of each sensor channel are normalized to lie within the range [0, 1].
* Following [11], the IMUs recordings are down-sampled to 30 Hz. This is a necessary step in ensuring that the sliding window size, *T*, used by [11] can be implemented by this project. For example, [11] uses a window size of 3 seconds or *T* = 100. But because the sampling frequency of the IMUs is 100 Hz, a window size of 3 seconds in the raw data corresponds to *T* = 300 samples. Therefore, the raw data must be down-sampled to 30 Hz. Now a window size of roughly 3 seconds can be achieved by letting *T* = 100.

After all of the participant’s files have been processed, there now exists training, validation and test data matrices along with their corresponding label vectors. These are placed in a pickle file which is output by the pre-processing script. This pickle file can be saved to the location of any path as input by the user. [11] makes publicly available their pre-processing script, written in python 2. This project adopts the script used by [11], updating and editing it so that it may be run successfully in python 3.

## 4.3 Baseline Approach Implementation

As described in chapter 3.2, two architectures; CNN-baseline and CNN-IMU, which have only minor differences, are implemented. Implementing a simpler CNN-baseline architecture is done for two reasons. The first is to replicate the benchmark paper, [11]. Secondly, it is also more sensible to begin with the more ubiquitous approach of a single-branched CNN architecture before progressing this to a CNN-IMU architecture with separate branches per IMU. In this section, the implementation of both models is described. First, the implementation details which are common between the two models are described. Then the implementation details of the specific architectures and data-loaders are described for each model.

### 4.3.1 Implementation Details

For both of the baseline architectures, the following implementation details are followed. This is a summary of the key parameters and structures used for the implementation of the baseline approach as described in 3.2. The rest of 4.3 gives a more in-depth analysis of the steps required to implement a baseline approach and successfully train a model.

**The Inputs**

The input images are generated using a sliding window approach with a window size of 3 s or *T* = 100 and a frame shift of 660 ms or *s* = 22. The small step size allows for a 78% overlapping and helps to generate a large number of input segments [11]. The sequence segments fed to the networks are labelled with their most frequent ground truth [11]. Sliding window method 1 is used to generate the inputs to

**The Architectures**

Although the CNN-baseline model consists of just one branch to process the data from all IMUs together whereas the CNN-IMU processes the data from each IMU individually, the architecture used in a single branch remains the same between models. This architecture takes a single-channel 2D *virtual* image (sequence segment from sliding window) of the sensory data as input. For each branch, this project implements *two* blocks. Each block consists of two stacked [5 x 1] temporal convolutions followed by a [2 x 1] max pooling operation. The output of the second block is flattened and passed to a fully connected layer. Two subsequent fully connected layers are used. The first two fully connected layers have 512 neurons each and the final fully connected layer has *K* = 12 logits. The activation function used on the output of each convolution operation and fully connected layer (excluding the classification layer) is the ReLU activation function. The output layer uses a softmax activation to calculate the pseudo-probabilities from the *K* class scores.

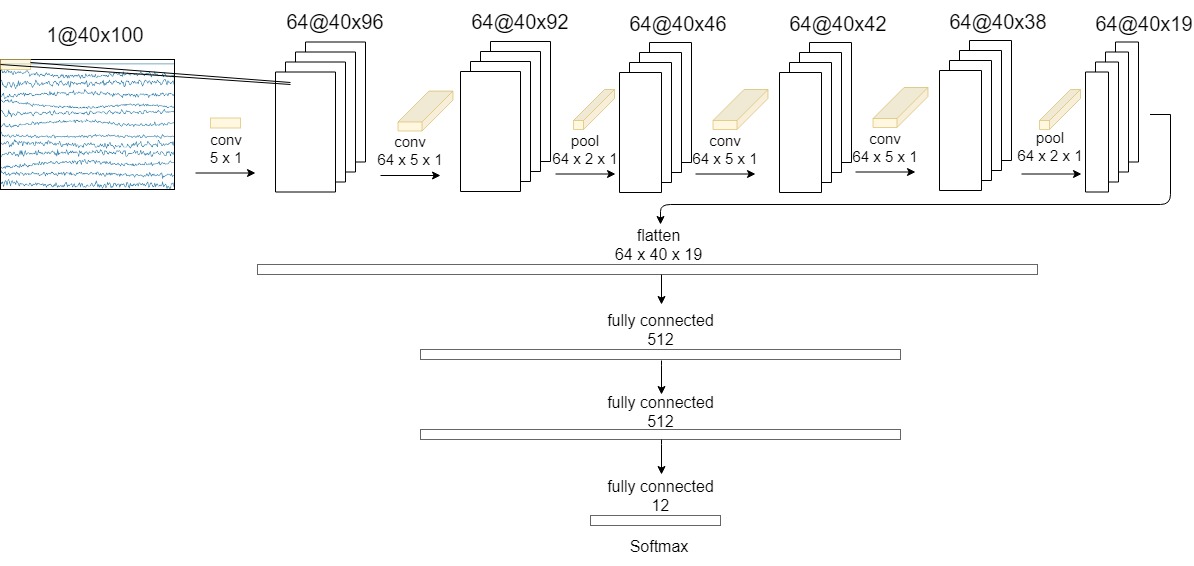
**Training**

When training the network, the network parameters are updated by minimizing the cross-entropy loss between the estimated probabilities, , and the target labels, [11]. This is done by using stochastic gradient descent with the RMSProp update rule [11]. The following hyperparameters are applied: RMS decay (*α* – smoothing constant) of 0.95, base learning rates of *β* = [, ] and a batch size of 50 [11]. The models in the baseline approach are trained for 20 epochs.

### 4.3.2 The Architectures: CNN-2 & CNN-IMU-2

#### CNN-2

The CNN-Baseline architecture with *B* = 2 blocks is illustrated below:



As the architecture illustrated above is the basic architecture on which the baseline approach as well as the experiments are based, a more detailed description of the parameters used in this model is given below:

**Block 1:**

The input *virtual* single-channel image is passed to a 2D convolution function. The image is convolved using a [5 x 1] kernel with a stride of 1, and 0 padding. The number of filters used is 64. Therefore, this convolution outputs 64 feature-maps. The feature-map images output from the previous convolution are passed through a rectified linear unit (ReLU) activation function. Local response normalization is then applied. 5 neighbouring channels are used for normalization. The multiplicative factor (α), exponent (β) and additive factor (*k*) are left to their default values of 0.0001, 0.75 and 1.0 respectively.

A second convolution is applied, this time where the input has 64 channels. All other parameters remain the same. A ReLU activation is applied once again. Local response normalization is applied using the same parameters. This block finishes with a 2D max pooling function using a kernel of size [2 x 1] and a stride of 1.

**Block 2:**

This block is very similar to the previous, with only slight adjustments. It consists of two stacked 2D convolutions where the inputs have 64 channels. After each convolution, a ReLU activation function is applied. Max pooling is applied followed by dropout of 50%. The parameters used for the convolution and pooling operations remain unchanged from block one. No local response normalization is used in this block.

**Fully Connected Layer 1:**

As can be seen by the figure axxxa, the input to the first fully connected layer is the output of block two after being flattened to one dimension. The convolution and pooling operations of blocks one and two cause the output feature maps to be of size 40 x 19. Therefore, the length of the input to the first fully connected layer is: 64 x 40 x 19. This flattened tensor is fully connected to a layer with 512 neurons. The output of the first fully connected layer is passed through a ReLU activation function before dropout of 50% is applied.

**Fully Connected Layer 2:**

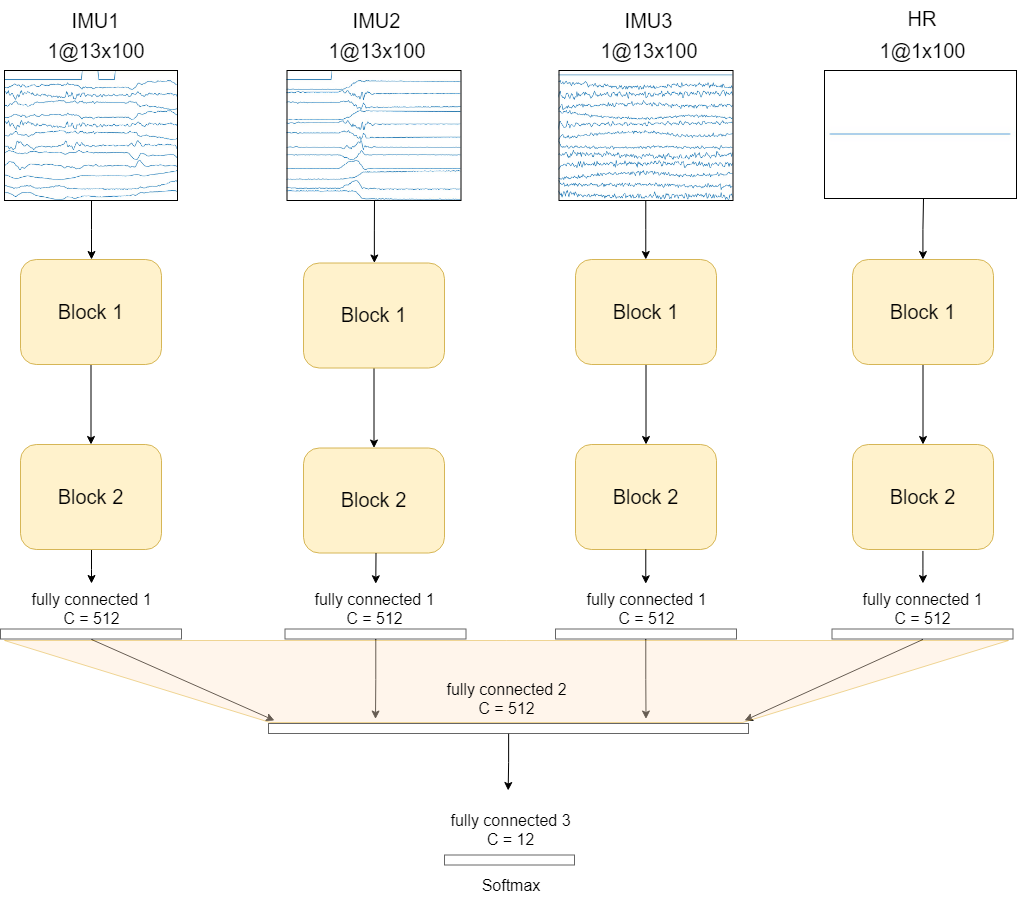
The output of the previous layer (512 neurons) is fully connected to another layer with 512 neurons. The output of this fully connected layer is passed through another ReLU activation function.

**Fully Connected Layer 3:**

The output of fully connected layer 2 is fully connected to a final layer with *K* = 12 neurons (an output neuron for each class). This is the final layer of the model, which is passed to a softmax classifier when training.

#### CNN-IMU-2

The CNN-IMU-2 (where *B* = 2 blocks) architecture is very similar to the CNN-2 above. The only difference is *m* = 4 branches are used, each processing segments from a specific IMU (or HR-monitor). To merge these branches together, the outputs of fully connected layer 1 at the end of each branch are concatenated before joined to the second fully connected layer. The following schematic illustrates this:



This is the only difference between the two architectures. To use either of these architectures for training, the model simply needs to be instantiated and then moved to the GPU. Once this is done, the model can be passed to the training function along with the selected optimizer and epochs number. This is elaborated further in the training section 4.3.2. For all implementations in this project, the weights of an instantiated model are initialized by the default weight initializers in PyTorch.

### 4.3.1 Data Loading

The dataset to be used must be defined using the PyTorch dataset class. A data-loader, constructed with an object of the dataset class, is then used to load the items of data (in this case sequence segments) in batches of some specified batch size. As seen in 4.2, the training, validation and test data are saved to a pickle file. Note that these datasets are simply 2D arrays containing the processed sensory data measurements and not a list of segmented sequences ready to be passed straight into a network. The sequence segments to be passed to the network are generated when the data loader is constructed \*\*\*check.

In order to train a model, the dataset must be loaded and moved to the GPU. The sequence segments passed to the network are generated in real time by the data-loader. Each time the data-loader requests an item from the dataset class, the dataset class shifts along the temporal axis by *s* samples (sliding window approach) before returning a segment of the dataset (of size [*T*, *D*]). A batch size of 50 is used [11], meaning the data-loader returns 50 images and their labels each time it is called.

When training the model, the training and validation datasets are constructed using the training and validation data saved to this pickle file. When the training/validation data-loader is constructed to load the batches of data samples, the data-loader calls the “\_getitem\_” method of the dataset object it was constructed with for each item of data in the dataset. This method returns a single indexed item of data at the specific index passed to the method. By overriding the PyTorch dataset class, the “\_getitem\_” method can be designed to implement the sliding window functionality. To implement the sliding window in the “\_getitem\_” method, the item index is multiplied by the frame shift, *s*, and the sequence segment of size [*T, D*] beginning at this new index is returned by the method. The number of windows that should be generated (i.e. the number of times this method should be called) equates to the following: , where *M* is the number of measurement samples in the data.

the PyTorch dataset class is used to load the correct dataset into the program. It must also be defined how a single input item is extracted from the dataset. To do this, the Dataset class must be overridden. In this case, the dataset class is overridden so that its constructor accepts a pickle file (containing training, test and validation datasets), a boolean training argument, a window size *T*, and a frame shift *s*. The boolean training argument allows the constructor to correctly decide which data to extract from the pickle file passed, i.e. training or validation. The window size *T* and frame shift *s* are needed by the “getitem” method of the dataset class, which is responsible for returning one sample of data (in this case a sample of data refers to one window, or *virtual* image, of sensor data). Each time the “getitem” method is called by the data-loader via the dataset class, the windowing function slides by a step of *s* and returns a segment of size [*T, D*].

**Data-Loading for CNN-2**

For the implementation of the CNN-2 architecture, the training and validation data-loaders used return one sequence segment of size [*T* = 100*, D* = 40], along with the most frequent label across the *T* = 100 data samples, as a single data item. This is done using sliding window method 1.

**Data-Loading for CNN-IMU-2**

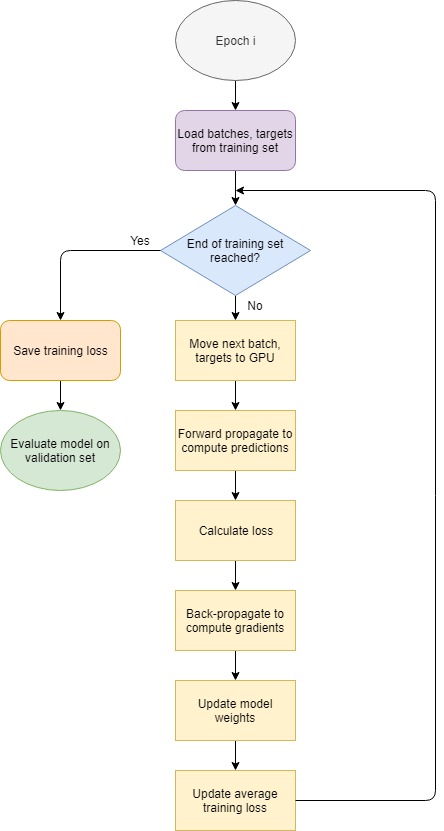
For implementation of the CNN-IMU-2 architecture, the training and validation data-loaders used return *m* = 4 sequence segments, each of size [*T = 100*, = 13], [*T = 100*, = 13], [*T = 100*, = 13] and [*T = 100*, = 1]. The most frequent label across the *T* = 100 data samples of these sequence segments is also returned. To summarize, when a data-loader requests one data item from the dataset, five objects are returned: *m* = 4 input sequences and a label. Sliding window method 2 is used to generate these input *virtual* images.

### 4.3.2 Training

**Training**

The training function runs for 20 epochs. Each epoch, the program iterates through the batches in the training data-loader. For each batch in the data-loader, the following occurs. The batch and their respective labels are moved to the GPU. The batch is fed to the model, which returns scores for each of the *K* classes. The softmax activation function computes the probability distribution of these class scores. The cross-entropy loss between the probability distribution of the model’s predictions and the true probability distribution (defined by the label) is found. The gradients of this loss are then computed so that the optimizer can update the weights of the model accordingly.

Each epoch, the training loss is recorded. The training loss for one epoch is calculated as the average cross-entropy loss of each batch. Therefore, the loss calculated for this batch is also used to update the average training loss for the current epoch. When all the training data has been fed to the model, the model is said to have been trained for one epoch. The model is then evaluated on unseen data using the validation set. The following figure illustrates how this project trains a model for one epoch:



**Validation**

When the network has been trained for one epoch, it is evaluated on the validation set to estimate how to current model might do on unseen data. Just as done for training, batches of the validation set are loaded using the validation data-loader and passed to the model. For each batch, the cross-entropy loss between the estimated probabilities and the target label is found. The loss calculated for this batch is used to update the average loss across the entire validation dataset for the current epoch. The predicted labels returned by the model for each batch are also saved. Therefore, at the end of an epoch of the validation set, a vector holding the predicted label for each sample of data in the validation set has been accumulated. At the end of the current epoch, the predicted labels are then compared against the true labels to calculate the classification accuracy for the validation dataset.

The training of the CNN-2 and CNN-IMU-2 architectures work in the exact same way. For the CNN-2 network, a single batch and label will be returned by the data-loader and fed to the model. For the CNN-IMU-2 network, *m* = 4 batches and a label will be returned by the data-loader and passed to the model.

### 4.3.3 Errors

4.3.2 mentions that it is important to keep track of the loss when training a model. However, the type of average loss is not mentioned. This project uses an exponentially decayed moving average to keep the track of the training loss each epoch. Exponential smoothing is used to combat noisy losses on individual batches so that trends become more visible.

When evaluating the model on the validation set, it is the average loss across the entire validation dataset that is needed. Hence, a cumulative running average is used to keep track of the validation loss.

### 4.3.4 Evaluation Metric

### 4.3.5 Testing

Evaluating the network on the validation set each time it has been trained for one epoch aids in the fine tuning of learning parameters for better training of the network. A drawback of this is it creates bias in the model toward the validation set as the learning parameters are tuned to fit the validation set. Hence, after the network has been trained for 20 epochs, this project saves the model and tests it on the test dataset. This is done to give an indication as to how the trained network will perform on new, unseen data in the real world.

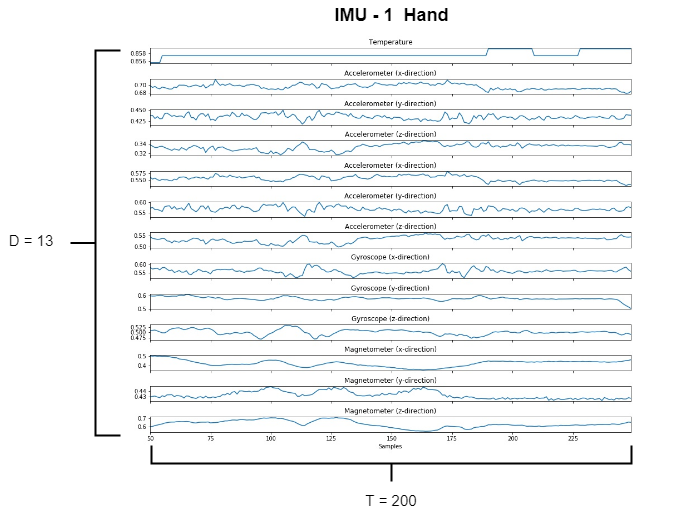
This project trains

## 4.4 Implementation of Experiments

The implementation of the experiments differs to the baseline approach in three ways. Firstly, the inputs images are real JPEG images and not *virtual* images. Secondly, these input images are not generated in real time as the model trains, i.e. an image is not plotted when an image is requested by the data-loader. These images are generated and stored in persistent memory prior to training/testing a model. These input images are retrieved as needed from their location in memory during training/testing by the data-loader. Finally, the kernels used for convolution and pooling operations on the image data are 2D kernels (e.g. [5 x 5]), as opposed to the 1D (e.g. [5 x 1]) kernels used in the baseline approach. These experiments use the CNN-IMU architecture developed in the baseline approach.

### 4.4.1 Machine Vision Approach 1

As described in the design section, this approach is very similar to the baseline CNN-IMU approach. The key difference here is the modality transformation of the input. Instead of passing a *virtual* image to the network, an actual JPEG image is passed instead. The image passed into each of the respective IMU branches contains the plots of each of the sensors of the respective IMUs. These images are plotted using Matplotlib. The concept of a sliding window is still applied here. The difference is that instead of passing the 2D array of sensor channels and *T* measurement samples to the network, an image of *D* subplots (a subplot for each sensor) is passed where each sensor channel is plotted for *T* measurements. The example below shows how a single input image for IMU 1 is generated where the window size, *T*, is 200:



Each time the window is shifted by *s*, three images must be plotted, one for each IMU. A python script is written to generate these input images for each of the IMUs, for a given dataset (train, validation or test). The images for each IMU are stored in separate directories. Each time an image is saved to each of the IMU directories, the figure index and label are saved to a csv file. The following flow diagram illustrates how the training dataset is created using this approach:

This new input type is then used to train a model using the CNN-IMU architecture as done in the baseline approach.

This is then integrated into the baseline approach which uses the CNN-IMU architecture.

### 4.4.2 Machine Vision Approach 2

### 4.4.3 Machine Vision Approach 3

# Chapter 5 - Results and Discussion

This chapter analyses the performances of each of the networks described in the implementation section. The highest validation accuracy and weighted F1 scores obtained by the models are displayed, along with their test results. 5.1 compares the results obtained in the baseline approach of this project to the results presented by the benchmark paper, [11]. 5.2 compares the results obtained by the novel approaches to HAR described in 4.4. Finally, 5.3 discusses the results of the novel machine vision approach to HAR with respect to the results obtained by the baseline approach of this project.

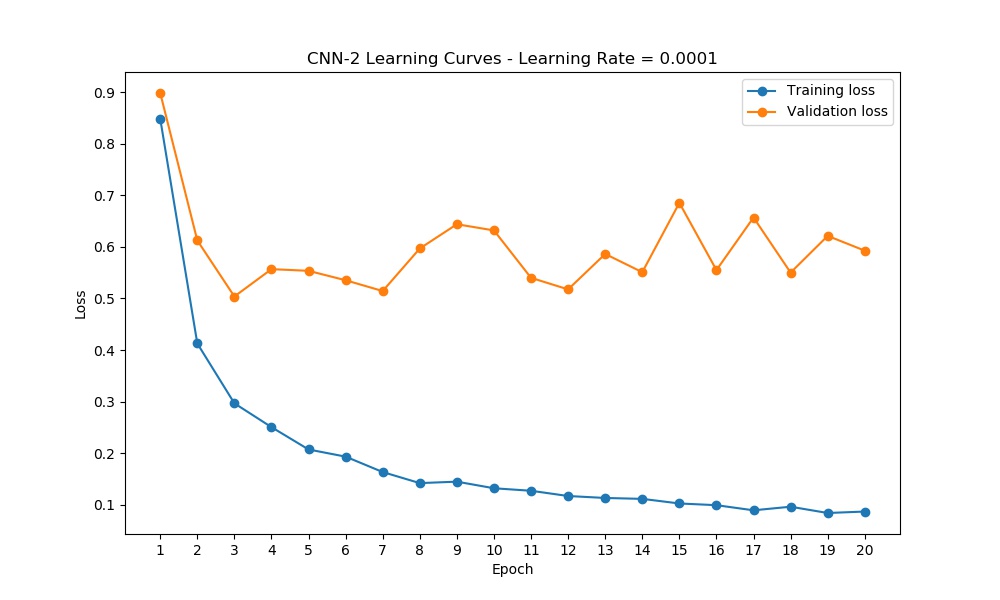
## 5.1 Baseline Approach

The following table shows the validation accuracies and weighted F1 for the CNN-2 and CNN-IMU-2 architectures obtained by [11], and then obtained by this project:

|  |  |  |  |
| --- | --- | --- | --- |
| *Validation* | Architecture | Acc (%) | wF1 (%) |
| Benchmark |  |  |  |
|  | CNN-2 | 91.15 | 91.22 |
|  | CNN-IMU-2 | 91.22 | 91.25 |
|  |  |  |  |
| Project |  |  |  |
|  | CNN-2 | 92.10 | 92.00 |
|  | CNN-IMU-2 | 88.99 | 88.65 |

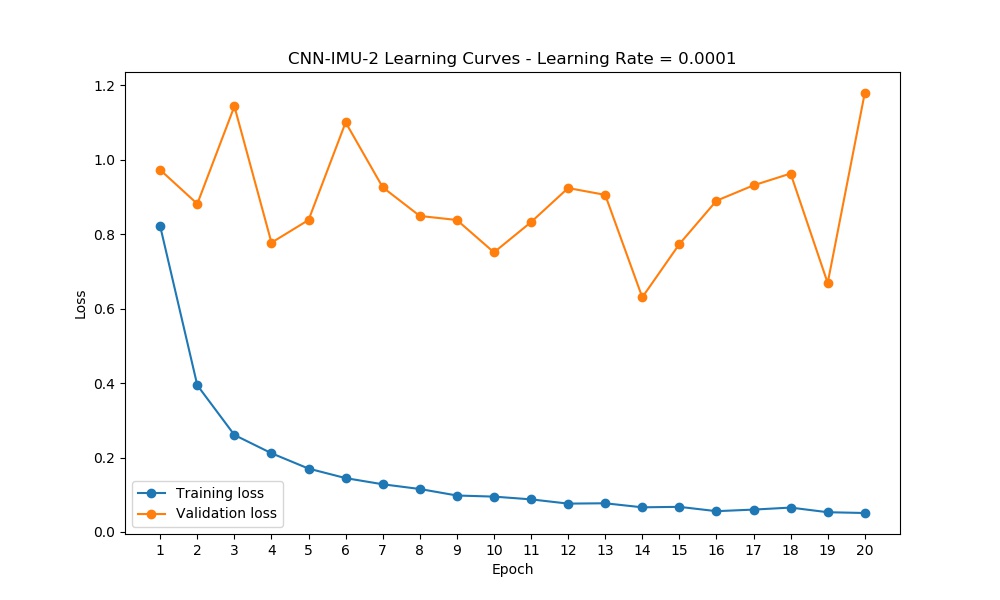
In this project, the best validation results were found using a learning rate of . As can be seen above, the CNN-2 baseline architecture outperforms the novel CNN-IMU-2 architecture proposed by [11] on the validation set. Although this project was implemented using a different framework (PyTorch) to that used in [11] (Caffe), this project achieves similar validation accuracy and weighted F1 scores for the CNN-2 and CNN-IMU-2 architectures as found by [11]. The discrepancy between the results of this project and the benchmark paper demonstrates the difficulty in reproducing a paper exactly and can exist for several reasons. The default parameters of modules can often vary between deep learning frameworks. It is also possible that [11] did not include all of the implementation details used when fine tuning their networks. With that said, the results found by the architectures used in this baseline approach serve as a good benchmark for the experiments carried out by the rest of this project.

The following learning curve shows the training and validation costs vs. epochs when training the CNN-2 baseline using the Pamap2 dataset:



As can be seen above, the network starts to overfit the training set after only a few epochs, where the validation cost converges to around a 0.6 loss.

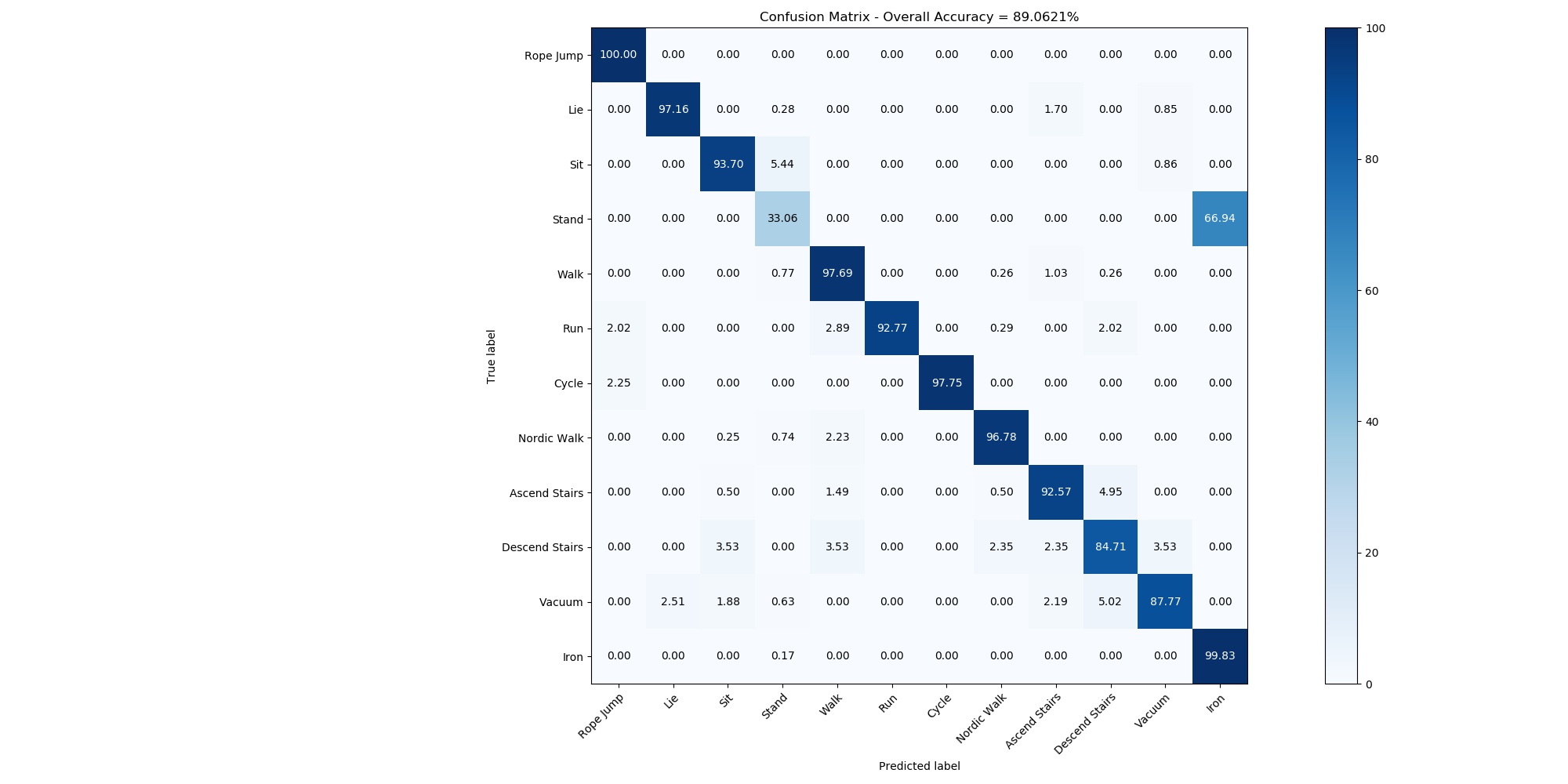
The CNN-IMU-2 network shows similar overfitting, with a validation loss which has larger fluctuations:



The CNN-2 and CNN-IMU-2 models which produced the validation accuracy and weighted F1 scores seen in table xnumx are then tested on the test set. They are tested on the test set to estimate the out-of-sample-error of the trained networks. The results produced on the test set are displayed on the following table:

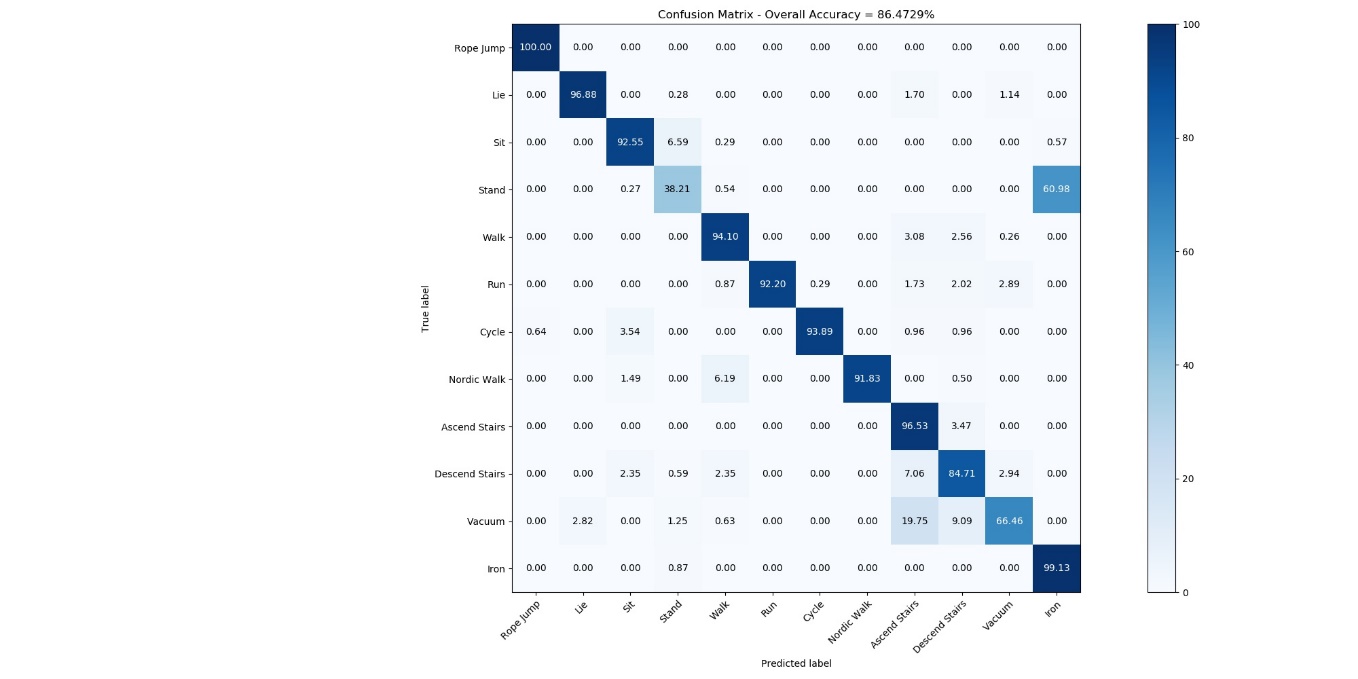
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Test* | Architecture | Acc (%) | wF1 (%) | Error |
| Project | CNN-2 | 89.06 | 88.27 | 0.6240 |
|  | CNN-IMU-2 | 86.47 | 85.82 | 0.6378 |

Although there is nothing to compare these test results against in the benchmark paper, [11], by comparing them to the validation accuracy and weighted F1 scores displayed in table xnumx it can be concluded that the trained CNN-2 and CNN-IMU-2 networks generalise quite well on unseen data. The following confusion graph shows how well the CNN-2 architecture classifies the *K* = 12 activities:



The largest source of error comes from being unable to distinguish standing from ironing. The misclassification rate of standing for ironing on the test set is 66.94%. The classification rate of the other eleven activities is quite high for each, with nine activities achieving a classification accuracy of above 90%.

The CNN-IMU-2 network achieves a marginally higher classification rate for standing, but is much less competent at classifying vacuuming correctly, with an accuracy of 66.46%.



## 5.2 Machine Vision Approach

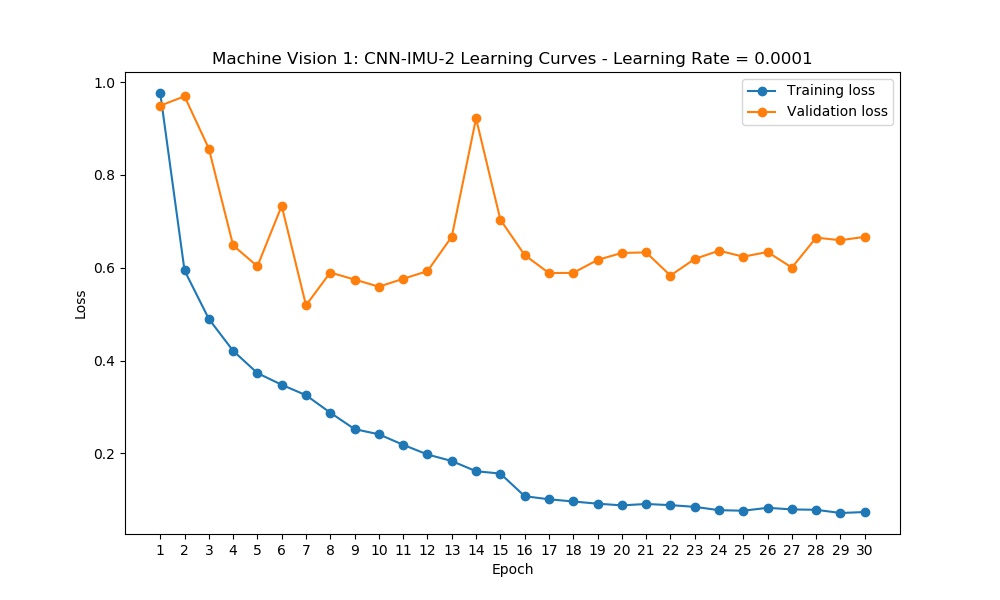
The machine vision approaches 1-3 produce varying results. Approaches 2 and 3 achieve 75.0242% and 79.5829% as their strongest validation accuracies and so are not competitive with the baseline networks. It is clear that although machine vision approach 3 aims to use a much richer image as input than those generated in the first two experimental approaches, the loss in the finer detail of the signals due to the sharing of subplots amongst signals proves to negatively affect the performance.

Experiments 1-a and 1-b both produce strong results competitive with those produced by the baseline approach. Different learning rates are experimented with when training these networks. They are also trained for 30 epochs so that learning rate reductions can be experimented with. For both 1-a and 1-b, the best results are found when the models begin training with a learning rate of and are decreased with at half the epochs. These results are displayed in the table below:

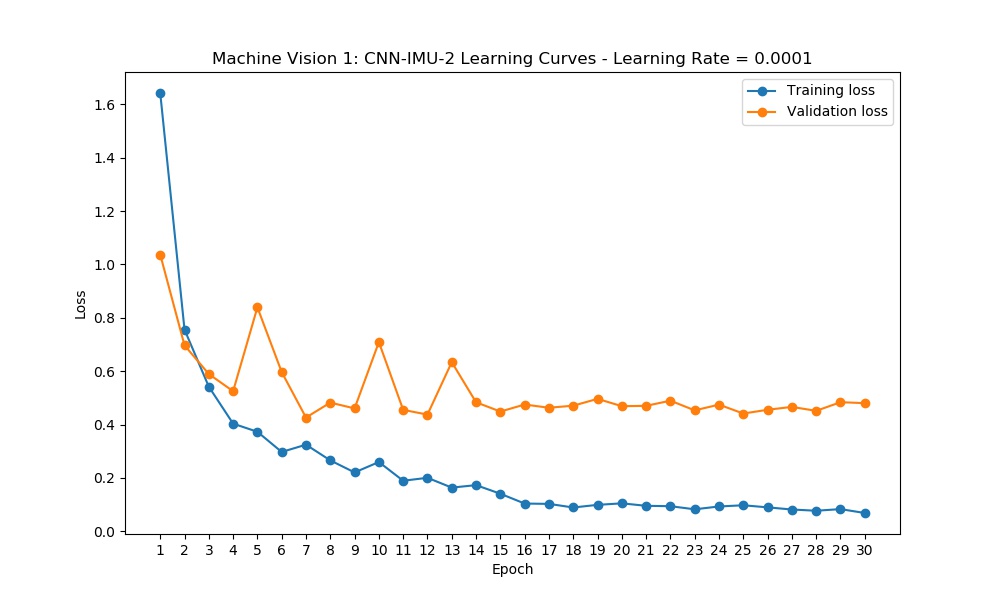
|  |  |  |  |
| --- | --- | --- | --- |
| *Validation* |  | @ |  |
|  | **Architecture** | **Acc (%)** | **wF1 (%)** |
|  | 1-a CNN-IMU-2 | 85.09 | 85.25 |
|  | 1-b CNN-IMU-2 | 87.70 | 87.60 |

Observed from the results above is a higher accuracy and weighted F1 score for 1-b. Following [15], machine vision approach 1-b generates the inputs to the CNN-IMU network using a larger window size of *T* = 200 (~ 6 s) and a larger step size of *s* = 50 (~ 1.5 s). Machine vision approach 1-a generates the input images using the original values of *T* = 100 and *s* = 22 from the baseline approach. The stronger results for network 1-b suggest that capturing an activity for a longer period of time using a larger window size *T* provides a more valuable input image. The learning curves for both approaches are seen below.

**Machine Vision Approach 1-a: *T* = 100, *s* = 22**



**Machine Vision Approach 1-b: *T* = 200, *s* = 50**

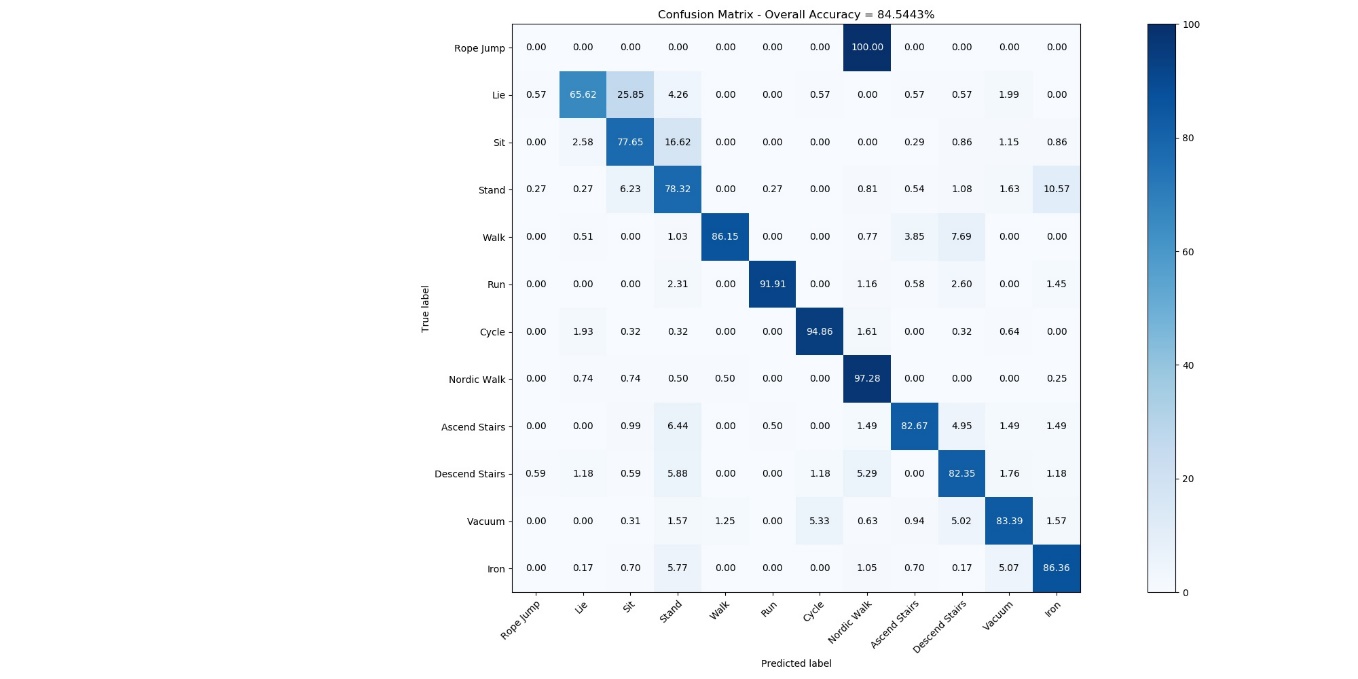


1-a initially appears to learn quicker than 1-b, but with a high validation loss. The validation loss converges to a steadier value of ~0.6 when the learning rate is reduced to at half the epochs. The validation cost of 1-a fluctuates a lot before the learning rate is reduced to . The validation cost of 1-a converges to a steady loss of ~0.5.

Machine vision approaches 1-a and 1-b which produced the validation accuracy and weighted F1 scores seen in table xnumx are then tested on the test set to estimate the out-of-sample-error of the trained networks. The results produced on the test set are displayed on the following table:

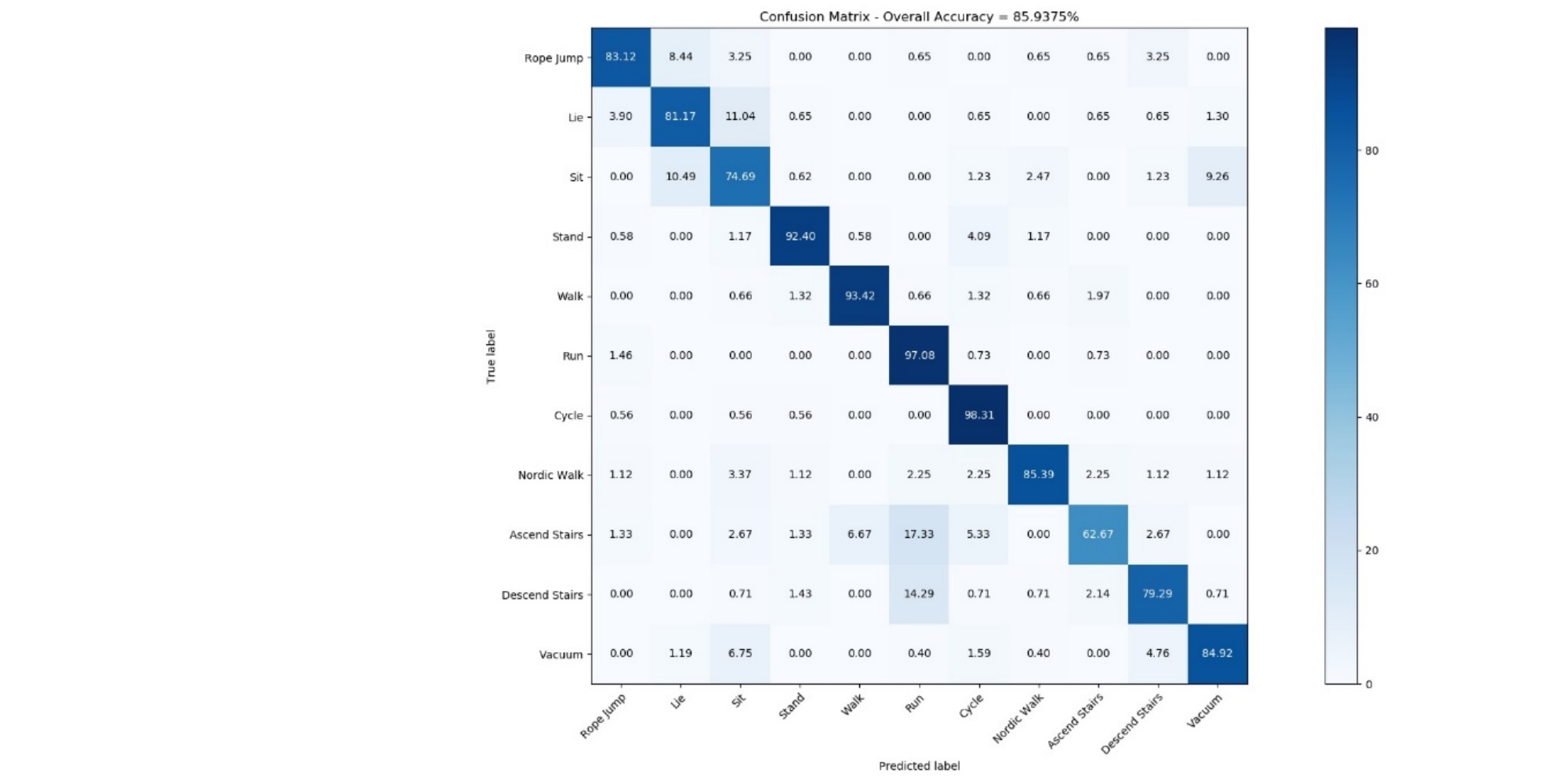
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Test* | Architecture | Acc (%) | wF1 (%) | Error |
|  | 1-a CNN-IMU-2 | 84.54 | 84.79 | 0.5676 |
|  | 1-b CNN-IMU-2 | 85.94 | 85.88 | 0.4643 |

The test results for both 1-a and 1-b show good generalisation, performing similarly well on unseen data as they do on the validation set. The following confusion graph shows how well the 1-a CNN-IMU-2 architecture classifies the *K* = 12 activities:



Quite interestingly, 1-a is far better at classifying standing with a classification accuracy of 78.32%. This is a big jump from the 33.06% and 38.21% classification accuracies for standing produced by the baseline CNN-2 and CNN-IMU-2 models. Although good all-round classification accuracy for most of the *K* = 12 classes, only three activities have a classification accuracy above 90% as compared to nine for the baseline CNN-2 and CNN-IMU-2 networks. 1-a also achieves a 0% classification accuracy for rope jumping.

The following confusion graph shows the classification accuracies for each of the *K* = 12 classes when the 1-b CNN-IMU-2 network is tested on the test set:



1-b improves the classification accuracy of standing to 92.40%, which is by far the highest seen from both the baseline networks and 1-a. Network 1-b also improves greatly on 1-a regarding the classification accuracy of rope jumping, with 83.12% compared to 0%. Where 1-b has most trouble is the misclassification of ascending/descending stairs with running.

## 5.3 Discussion

After testing the two baseline architectures and both 1-a and 1-b on the test set, it is apparent that each have their strength and weaknesses when it comes to the type of activity they try to classify. The baseline CNN-2 and CNN-IMU-2 architectures, although do not produce exactly the same results, are both very strong at classifying the majority of the *K* = 12 activities in the Pamap2 dataset. Their downfall, however, is the classification of standing, confusing it with ironing over 60% of the time. 1-a and 1-b do not have the same problem, with 1-b classifying standing correctly 92.4% of the time. The trade-off with this approach is its trouble classifying ascending and descending stairs correctly, with accuracies of 62.67% and 79.29% respectively. 1-a also has a good all-round classification of the *K* = 12 activities, but misclassifies rope jumping with Nordic walking 100% of the time. 1-b classifying rope jumping with 83.12% accuracy demonstrates the impact parameters *T* and *s* have on training a model.

The following table shows the test results for the baseline approach and machine vision approach 1 alongside each other:

|  |  |  |  |
| --- | --- | --- | --- |
| *Test* | Architecture | Acc (%) | wF1 (%) |
| Baseline | CNN-2 | 89.06 | 88.27 |
|  | CNN-IMU-2 | 86.47 | 85.82 |
|  |  |  |  |
| Machine Vision 1 | 1-a CNN-IMU-2 | 84.54 | 84.79 |
|  | 1-b CNN-IMU-2 | 85.94 | 85.88 |

The CNN-2 baseline network has the highest overall performance. The 1-b CNN-IMU-2 network does, however, outperform the baseline CNN-IMU-2 network with a higher weight F1 score.

# Chapter 6 – Ethics

# Chapter 7 – Limitations, Conclusions and Further ResearchReferences

|  |  |
| --- | --- |
| [1] | O. C. Ann and L. B. Theng, “Human Activity Recognition: A Review,” in *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*, Batu Ferringhi, Malaysia, 28-30 Nov. 2014. |
| [2] | J. Bo Yang, M. N. Nguyen, P. P. San and X. K. S. Li Li, “Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition,” in *Twenty-Fourth Joint Conference on Artificial Intelligence 2015*, 2015. |
| [3] | M. S. Ryoo, “Human activity prediction: Early recognition of ongoing activities from streaming videos,” in *IEEE International Conference on Computer Vision (ICCV)*, Barcelona, Spain, 2011. |
| [4] | W. S. Lasecki, Y. C. Song, H. Kautz and J. P. Bigham, “Real-time crowd labeling for deployable activity recognition,” in *2013 conference on Computer supported cooperative work-CSCW 13*, San Antonio, Texas, 2013. |
| [5] | E. S. Sazonov, G. Fulk, J. Hill, Y. Schutz and R. Browning, “Monitoring of posture allocations and activities by a shoe-based wearable sensor,” *IEEE Transactions on Biomedical Engineering,* vol. 58, no. 4, pp. 983-990, 2011. |
| [6] | G. Paragliola and A. Coronato, “Intelligent Monitoring of Stereotyped Motion Disorders in Case of Children with Autism,” in *2013 9th International Conference on Intelligent Environments - IEEE*, Athens, 2013. |
| [7] | E. Kańtoch and P. Augustyniak, “Human activity surveillance based on wearable body sensor network,” in *2012 Computing in Cardiology*, Krakow, 2012. |
| [8] | E. E. Stone and M. . Skubic, “Passive, in-home gait measurement using an inexpensive depth camera: Initial results,” , 2012. [Online]. Available: http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000006240383. [Accessed 21 3 2019]. |
| [9] | A. T. Nghiem, E. . Auvinet and J. . Meunier, “Head detection using Kinect camera and its application to fall detection,” , 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6310538. [Accessed 21 3 2019]. |
| [10] | J. Wang, Y. Chen, S. Hao, X. Peng and L. Hu, “Deep Learning for Sensor-based Activity Recognition: A Survey,” *Pattern Recognition Letters, 2019 - Elsevier,* no. 119, pp. 3-11, 2019. |
| [11] | F. Moya Rueda, R. G. Grzeszick, G. A. Fink, S. Feldhorst and M. ten Hompel, “Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors,” *Informatics,* Vols. Informatics 2018, 5, 26, 2018. |
| [12] | M. Singh, V. Pondenkandath, B. Zhou, P. Lukowicz and M. Liwickit, “Transforming sensor data to the image domain for deep learning—An application to footstep detection,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, 2017. |
| [13] | J. Yang, M. Nguyen, P. San, X. Li and S. Krishnaswamy, “Deep convolutional neural networks on multichannel time series for human activity recognition,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015. |
| [14] | M. Zeng, L. Nguyen, B. Yu, O. Mengshoel, J. Zhu, P. Wu and J. Zhang, “Convolutional neural networks for human activity recognition using mobile sensors,” in *6th International Conference on Mobile Computing, Applications and Services*, 2014. |
| [15] | E. Brophy, J. Dominguez, Z. Wang and T. Ward, “A machine vision approach to human activity recognition using photoplethysmograph sensor data,” in *2018 29th Irish Signals and Systems Conference (ISSC)*, 2018. |
| [16] | N. Hammerla, S. Halloran and T. Plötz, “Deep, convolutional, and recurrent models for human activity recognition using wearables,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016. |
| [17] | S. Ha, J. Yun and S. Choi, “Multi-Modal Convolutional Neural Networks for Activity Recognition,” in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015. |
| [18] | W. Jiang and Z. Yin, “Human activity recognition using wearable sensors by deep convolutional neural networks,” in *23rd ACM international conference on Multimedia*, 2015. |
| [19] | S. Ha and S. Choi, “Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, 2016. |
| [20] | A. Reiss and D. Stricker, “Introducing a New Benchmarked Dataset for Activity Monitoring,” in *The 16th IEEE International Symposium on Wearable Computers (ISWC)*, 2012. |
| [21] | “UCI Machine Learning Repository,” 2007. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/pamap2+physical+activity+monitoring. [Accessed January 2019]. |

# Appendix 1

# Glossary