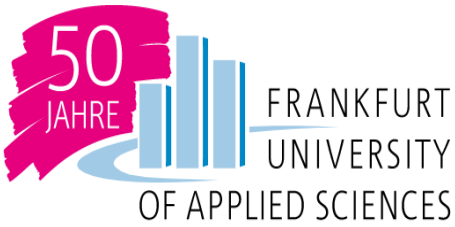

Master Thesis

Pre-training Deep Architectures for Online Time-Series Classification

A thesis submitted in fulfillment of the requirements for
the degree of ***Master of Science (M.Sc.) in High Integrity Systems (HIS)***

to the

Frankfurt University of Applied Sciences
Department of Computer Science and Engineering

by

Kshitij Yelpale
Matriculation No.: 1322509
kshitij.yelpale@stud.fra-uas.de

Academic Supervisor
Prof. Dr. Ute Bauer-Wersing
ubauer@fb2.fra-uas.de

Industrial Supervisor
Dr. Viktor Losing
viktor.losing@honda-ri.de

January 03, 2022
Frankfurt am Main

Declaration of Authorship

I, Kshitij Yelpale, declare that this thesis titled **"Pre-training Deep Architectures for Online Time-Series Classification"** and the research work presented here are of my own. I confirm that:

- This work has been completed as a partial requirement for my Master's degree.
- I have properly mentioned and cited other's published works used for this thesis.
- I have acknowledged all of the prime sources of help.
- I have always provided the source wherever I have quoted the work of others. Except for such quotations, this thesis is entirely my work.

Frankfurt, 03.01.2022

Place, Date

Signature

Abstract

The usage of pre-trained, machine-learning models has extensively increased nowadays by making it one of the most important topics of research. It is one of the most promising techniques to get the benefits of Deep Learning on a comparably small labeled dataset. However, the influence of pre-trained models is not uniform across all the deep neural networks. In this thesis, the potential usage of transfer learning for an interesting application scenario of users' motions has been investigated. The task is to classify individual user motion at a particular state of time. The data used in the research is collected by the Xsens Bodysuit, which gives many parameters of motion. The motions are in IMU-based Time-Series sequences form. The standard Deep Neural Networks (DNN) such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are applied. The pre-training has been performed on a similar dataset and fine-tuned with recorded data. Finally, it is then compared with normal training results. Considering a few parameters and the right hyperparameters, it has been concluded that pre-training is very useful to increase the performance of a model under certain hyperparameters settings when the data is scarce.

Acknowledgments

I would like to thank Prof. Dr. Ute Bauer-Wersing for allowing me to complete my master thesis at the Frankfurt University of Applied Sciences in cooperation with Honda Research Institute, Europe. Her constant trust and guidance helped me to keep up my pace in the completion. I am sincerely grateful to my supervisor at Institute, Ph.D. Viktor Losing, a Senior Data Scientist at HRI, for his ever-so-helpful attitude, constant support, and numerous fruitful discussions throughout the thesis, especially during the experiments. He always guided me towards the goal of the thesis.

Finally, I would like to thank my family and friends for their constant support and understanding throughout the thesis.

Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
Bi-LSTM	Bi-directional Long Short-Term Memory
CNN	Convolutional Neural Network
DLSTM	Deep Long Short-Term Memory
DTW	Dynamic Time Warping
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent unit
IMU	Inertial Measurement Unit
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
RNN	Recurrent Neural Network
TS	Time Series
TSC	Time Series Classifier
UCR	University of California, Riverside

Chapter 1

Introduction

Time-series classification has been received over the last few years which solved very interesting tasks where time is a main contributing factor. In the RT-Phume project at the research institute, the focus was on the task of online time-series classification and regression using different modalities such as IMU sensors, foot-sole pressure sensors, and gaze data. As the target is to achieve real-time performance, the plan is to record and process the data at a high-frequency rate. The motivation behind using pre-trained models in the motion time-series data is, that the increase in the performance improvement by just transferring relevant models either with fewer samples for fine-tuning or training gets completed in fewer epochs.

1.1 Motivation

The traditional machine learning and MLP models are performed well when there is output associated with input, but there are few tasks where inputs are dependent on each other in a progressive manner like sequence and outputs are limited in size and independent in an association such as machine translation, time-series. To capture this nature of data, modified deep learning techniques are needed like LSTMs which processes inputs in sequences and backpropagate the learning (gradients). Multivariate time-series data can be arranged in sequences. The length of the sequence is the time window and multiple columns (independent variables) processed together like embedding layer for words in NLP tasks.

Using machine learning algorithms and techniques, the performance of assistive devices can be improved by favoring the usage of ergonomics. The work conditions in the industry can be adapted with the latest technologies, such as collaboration with robots for workers' assistance, wearable devices to monitor the health situations and warn people in the case of emergencies. Because of improved predictions and human motion recognition, the collaborative robots enable safe interaction with the industry applications [1, 2] such as activity recognition module that identifies non-ergonomics gestures, learn surrogate models to learn the real-time posture of various tasks [3].

For higher efficacy and improvement in the performance of models, the data is divided into two datasets, training and testing. But sometimes the sophisticated

data is not enough to complete this process, then data is collected from the same domain but from different problem statements to create high-performance learners. It is motivated by consensus that transferring models between similar datasets improves classifiers accuracy [4]. This idea is very similar to the real world, where organizations hire employees who have more experience than new trainees or graduates. The previous experience is always easy to transfer and makes the task more accurate.

In the context of IMU-based data and motion classification, pre-training has rarely been used and available datasets are rather small, encouraging the application of pre-training. Therefore, the focus of this master thesis is the exploration and evaluation of pre-training methods in the domain of IMU-based motion classification.

1.2 Related Work

There always has been research papers published using deep learning neural networks in human activity recognition [5] and time-series classification [6] where multiple deep learning models are applied over different types of univariate and multivariate datasets. Authors have experimented with a variety of deep learning networks like CNNs and residual networks and categorized them into two parts, generative and discriminative models. The interesting finding done by authors is the black box effect of deep models uncovered by a Class Activation Map visualization which shows the parts of input time-series contributed to a particular class prediction [6].

Transfer learning can also be performed on time-series data using CNN, as shown by the authors in the paper [7], in which they tried UCR archive [8] 85 TS datasets and generated many fine-tuned networks. Using a method called Dynamic Time Warping to measure dataset similarities, transfer learning shows improvement on 71 out of 85 datasets. This DTW method compares datasets that are very closely related. The CNN techniques are inspired by the results from the paper [6] where the results are very successful in different domains. A bad choice of source dataset for a target dataset leads to the optimization algorithm getting stuck in a local optimum called negative transfer learning observed by authors [7]. Therefore, a big data practitioner should provide attention to the relationship between both datasets.

It is not always possible to have labeled dataset prepared for deep learning training. Therefore, unsupervised learning where unlabeled data which is commonly available in the most domain are used. The authors have developed a framework for unsupervised pre-training using the DLSTM network [9]. LSTM is used with autoencoder framework which solved the random weight initialization of DLSTM and successfully proposed a robust model trained on unlabeled multivariate time-series data which can be used for other tasks' analysis of time-series representations. The results showed that layer-wise pre-training improved the performance of DLSTM, with faster and better convergence.

Transformer attention models can also be used for handling the sequences. But experiments with multivariate data is not very common to analyze the results in the form of pre-training. Though, there are few articles available for single variate

time-series data. Some engineers experimented with multiple data types, and codes are available in GitHub [10–12].

1.3 Goal

The goals of this thesis are summarized as follows:

- Explore the possibilities of deep learning techniques to classify the motions using the bodysuit sensors data as sequences.
- Find the best hyperparameters in the process of transfer learning.
- Comparison between the classifiers for predicting the motions at a given time t.

The overall aim of this research is to evaluate the advantages of pre-training in the time-series data, which can be used to increase the performance of online learning. Though online learning outperformed the static average learning [13], this can give an advantage for personalized dynamic learning.

1.4 Contribution

The thesis has been organized into straightforward sections. Chapter 2 describes the background theory of Deep Learning related to basic concepts of neural networks and architectures of networks that are used in the experiments in the thesis, LSTM, and GRU. How the data has been collected and organized, is mentioned in chapter 3 Experimental Setup. Following that, the implementation is also included in the same. The next chapter 4 explains results which have very interesting facts about the experiments and interpretation are made. The last chapter 5 concludes the thesis and addresses future scope.

Chapter 2

Deep Learning

Machine learning is a subset of artificial intelligence in the field of computer science, which comprises the science of getting the task done by computers by providing the data and their results and generating a function or algorithm which can produce the results for the future data [14]. This approach is opposite to the methods used in computer science, in which the results are produced by input data and pre-defined algorithms. The mathematical function for this is expressed as follows.

$$y = f(x, w) \tag{2.1}$$

where x is the input data, w is the parameters learned and y is the output of the task. The applications of ML are large such as finding patterns, predicting a future event, classifying the objects, completion of occurrences, and many more. Since it is extremely diverse and complex, the chapter is very limited to the given problem.

2.1 Neural Network

Neural networks are representations of mathematical functions, that are inspired by biological neuron systems. Whereas, the direct comparison is still elusive. The artificial neural network depicts the operational concept of biological neurons' network. There is still deep research is going on to find out the exact representation to map with an artificial neural network and hence the human brain's neurons are so powerful. The ANNs are generally used to solve the problems of object detection, sentence completion, fraud detection, anomaly detection, and time-series relation forecasting, classification, and many more.

- **Artificial Neurons:** The artificial neuron is a representation of a biological neuron that acts as a central information processing component of the network, in which the neuron receives the signals through its dendrites from other neurons. The central cell processes it and sends it further to other neurons through its axioms. The synaptic gap between axioms and dendrites decides how much strong the connection is. The same mechanism also applies in ANN, where inputs are received to this processing unit called an artificial neuron. The weights

with inputs decide strong consideration of that input in the calculations. For example, if the weight is close to 0 then input is ignored. After collecting all the inputs, and summing up, an activation function has been applied to generate the result. McCulloch and Pits proposed a formal representation of artificial neurons in 1943 [15].

- **The Perceptron:** The basic unit of ANN is designed by Frank Rosenblatt to mimic human brain learning [16]. It is the first neural and smallest network with a single neuron initially introduced which is only capable of solving the linearly separable problems. It is represented as follows.

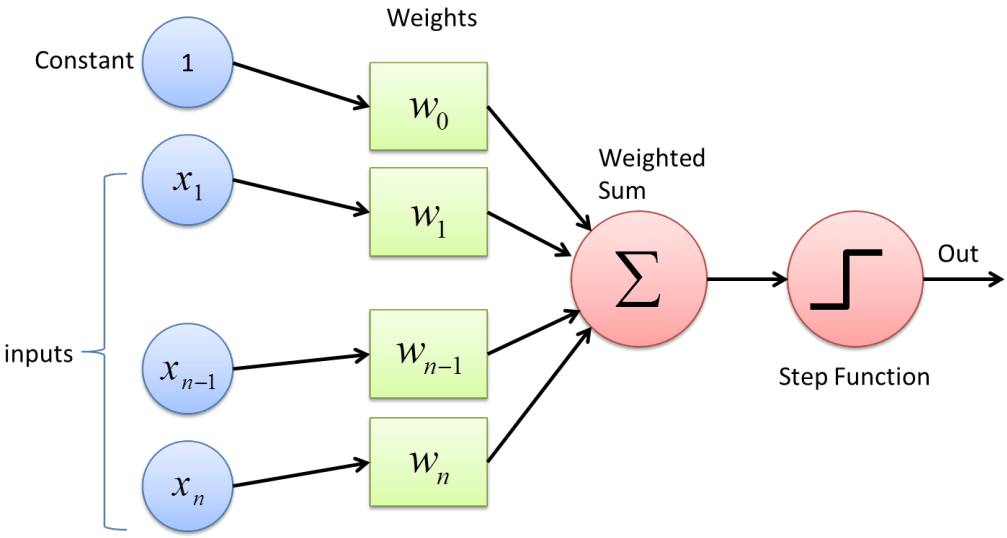

Figure 2.1: Perceptron
[17]

Perceptron is used for linear classification tasks and dwells in four parts – input values, weights and bias, summation function, and the activation function. The input x_1 through x_n are given to the perceptron and the output is computed using the following expression [18].

$$y = f(w_0 + \sum_{i=1}^{n} x_i w_i) \tag{2.2}$$

where w_0 is the bias and w_i are the weights associated with the input values and controls involvement in the computation. The expression can be simplified by adding the input x_0 as 1 for the initial, *bias* and it can be written as.

$$y = f(\sum_{i=0}^{n} x_i w_i) \tag{2.3}$$

The individual weights w_i and the bias w_0 can be chained into a weight matrix W and the incoming signals can be added into a matrix. [18]

$$\hat{x} = \begin{pmatrix} 1 \\ x \end{pmatrix} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix} and\, W = \begin{pmatrix} w_0 \\ w \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ . \\ . \\ w_n \end{pmatrix} \tag{2.4}$$

Combining the above two equations, the output y can be written as:

$$y = W^T x \tag{2.5}$$

To solve the non-linear separable problem, more than one neuron needs to combine and form a network. In which every neuron works the same as a perceptron but differs in the activation functions depending on the task. It is also called as Multi-layered Perceptron (MLP).

2.2 Neural Network Training

The problem of a single perceptron was solved by ANN [19]. It comprises multiple neurons present in its architecture. The structure is formed by the input layer, one or more intermediate layers, and output layer. The neurons from one layer are connected to the neurons from the next layer through weights and bias [20]. Now, the training process of neural networks should be discussed.

- **Learning Rate:** This is one of the important hyperparameters which can be used to tune the performance of the neural network. This is the step size in the training process by which the weights and bias are updated to learn the problem, to reduce the loss of the problem. If the value of the learning rate is high then the weight adaptation is large which results in a decrease in performance by wavering loss value over the training epochs [20]. Whereas if the value is to be chosen very small then the model takes an infinite amount of time to converge or get stuck at local minima. So, it is a monotonous task to choose the right value [21]. The value can vary between the range of 1 to 10^{-6} and the default value can be set as 0.01 [22]. This value can be updated during the training process.

- **Activation functions:** The mathematical functions are chosen to decide the output value for the given inputs. Many popular functions are useful for a specific task. Taking nonlinear decisions are made possible by activations functions. The different activations are as follows,

  1. Identity function

$$f(x) = x \tag{2.6}$$

2. Sigmoid function

$$f(x) = \frac{1}{1 + e^{-gx}} \tag{2.7}$$

where g is a parameter to control steepness.

3. Hyperbolic tangent function

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.8}$$

4. Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} 0 & if\ x < 0 \\ x & otherwise \end{cases} \tag{2.9}$$

5. Step function

$$f(x) = \begin{cases} 0 & if\ x < 0 \\ 1 & otherwise \end{cases} \tag{2.10}$$

- **Loss functions:** Loss functions are also called cost functions which are crucial in the training process of neural networks. Through the training process, an ideal function needs to find out which maps the predicted outputs for the given inputs with the desired outputs. Loss function finds the difference between both the outputs. There are many loss functions used for the different problem tasks. The most used loss function in machine learning is Mean Squared Error (MSE) or L2 loss. The formula is as follows,

$$L_2 = \frac{1}{n} \sum_{i=1}^{n} L(y_i - \hat{y_i})^2 \tag{2.11}$$

where n is the number of inputs or data points, y_i is the expected output, and $\hat{y_i}$ is the predicted output by the model.

However, this loss function is not used with the sigmoid activation function because of slow convergence. Because the gradients become flat at the extreme points of sigmoid activation. Therefore, if the neurons are saturated at the wrong sides, the error will be large and the network could never converge or take more time to converge [20].

Cross Entropy Loss:

This is another loss function that is mostly used for classification tasks. It can be used with a sigmoid activation function. It gives relatively better results than MSE, converges faster, and can reach global minima.

$$L_{CE} = -\frac{1}{n}[y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})] \tag{2.12}$$

where n is the number of inputs or data points, y_i is the expected output and $\hat{y_i}$ is the predicted output by the model. In the experiments conducted for this thesis, this cross-entropy loss is used in all settings.

- **Gradient Descent:** Gradient descent is the optimization process in which the network parameters, i.e., weights and bias, are updated by minimizing the loss function L. The parameters are updated iteratively in the negative direction of the partial derivative of the loss function with respect to the parameters. The gradients show the direction towards the steepest ascent. The step size on each epoch or iteration is determined by the learning rate. At the start of the training process, the weights are initialized randomly and after each iteration, the partial derivative of the loss function is calculated by each parameter as:

$$\frac{\partial L(w)}{\partial w_{kj}}\bigg|_{W^T} = \Delta_{W_{kj}} L \tag{2.13}$$

$$\frac{\partial L(b)}{\partial b_{kj}}\bigg|_{b^T} = \Delta_{b_{kj}} L \tag{2.14}$$

  where w_{kj} and b_{kj} are the weight and bias connecting from the j^{th} neuron in layer l to k^{th} neuron in the next layer of l. T is the variable for iteration. The parameters can be updated using Stochastic Gradient Descent (SGD) as follows:

$$w_{kj}^{\tau+1} = w_{kj}^{\tau} - \eta \Delta_{w_{kj}} L \tag{2.15}$$

$$b_{kj}^{\tau+1} = b_{kj}^{\tau} - \eta \Delta_{b_{kj}} L \tag{2.16}$$

  where η is the learning rate

- **Backpropagation of error:** The authors proposed a backpropagation rule to update the parameters (weights and bias) exhibit in the hidden and output layers [20, 23]. The algorithm gained recognition after this successful and efficient training of the neural networks. Because of a fixed number of neurons, layers, and interconnected units, the backpropagation algorithm employs gradient descent to learn parameters by minimizing the loss function between the true and predicted outputs.

  There are other variants of SGD also used specifically for the problem statement it is required, such as batch gradient descent and mini-batch gradient descent. Mini-batch is mostly used because it splits the data into fixed batches, which ultimately decreases the load on memory and computational speed increases. Adagrad, adaptive moment estimation and RMSProp are also some other optimization algorithms. In this thesis, an Adaptive Moment Estimation (Adam) optimizer is used. This is an adaptive learning method algorithm and computes an adaptive learning rate for each parameter by exploring the first and second moments of the gradients.

2.3 Deep Learning Neural Networks

The need for deep learning was raised due to the inefficient performance of traditional machine learning algorithms on the complex representation of data or

sometimes the raw data processing, which also included unsupervised learning. Feature engineering is required when the traditional algorithms need to be used. But it is not a straightforward process and includes significant domain knowledge to design features. These networks can learn complex features automatically. Because there are multiple hidden layers stacked together along with nonlinear activation functions. The neurons from each layer receive the output of the neurons from previous layers. As the layer and neurons in hidden layers increases, the capacity to handle complex task increases. The special type of ANN is discussed further which are more useful to keep information in memory.

2.3.1 Recurrent Neural Network

Neural networks like ANNs and other machine learning methods cannot deal with the special type of data, which are sequences. Sequential data such as speech, language translation, video, and sensor data, time-series, etc., are dependent on the components which are bound by time. The plain networks process each input independently, and sequential information is lost while training. It can be handled in multiple ways, one is to concatenate sequential data in fixed numbers and consider them as one data point, just like a sliding window [24,25]. But it is difficult to find an ideal window size. Sequences as inputs for speech translation and machine translation could vary in length. RNNs can work with these sequential data [26]. Consider a sequence $x_1, x_2, \ldots, x_n$, and output is y_t, is calculated using the current hidden state h_t and using x_{t-1} at each time step t with the help of recurrent connections as follows [27]:

$$h_t = \sigma(b_h + W^{hx} x_t + W^{hh} h_{t-1}) \tag{2.17}$$

$$\hat{y_t} = softmax(b_y + W^{yh} h_t) \tag{2.18}$$

where b_h and b_y are the biases that provide as offset, W^{hx} is the weight matrix between the input and hidden layer, W^{hh} is the weight between hidden layers and W^{yh} is the weights between output and hidden layers. RNNs have the mechanism of storing the information present in the previous time-steps and nonlinearity updates the hidden states, which ultimately make these neural networks powerful. The Figure 2.2 explains the basic architecture of RNN. At all the time steps, the hidden state is shown as well as the output from the previous hidden state, and input at the current time step is given as input to the network that further outputs y at a time step with the hidden state. Due to feedback loops, each hidden state is dependent on the previous state and merges information from the previous states [20].

2.3.1.1 Backpropagation Through Time

The learning process is performed by Backpropagation Through Time (BPTT) [27]. This is just an extended version of the standard backpropagation algorithm, which is used in the training process of feed-forward neural networks. The standard

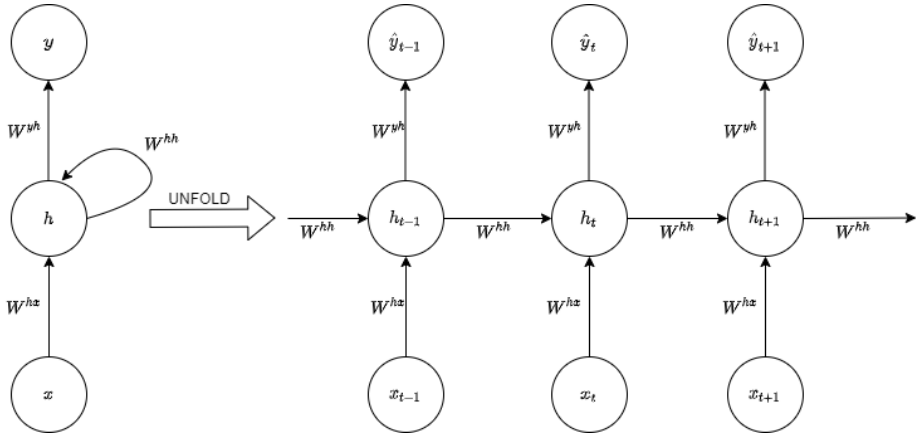

Figure 2.2: Recurrent neural network with three timestamps

algorithm cannot be used because of the cyclic graphs present in this architecture. Therefore, the network unfolded in time as shown in Figure 2.2 to train the network, which shows then the process like FNN [25]. Each time-step layer becomes parallel to the hidden layer in FNN when sharing the weights. Hence, the layers are connected next layers in time t to $t+1$.

2.3.1.2 The problem of Vanishing and Exploding Gradients

The major problem in RNNs is the vanishing and exploding gradients that are also available in other plain networks. But here it is challenging to handle, which is discussed further. The total error is computed by summing all errors at each time-steps based on the output $\hat{y}_t$ and the labels y_t. So, to minimize the error, the gradients are updated at each time step, and sum it is as follows [20, 27]:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \tag{2.19}$$

Now if the chain rule is applied to above two equations, then

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{2.20}$$

the above derivative of loss function E_t is shown, which depends on the hidden state activation h_t, and it is further dependent on all the previous hidden state activations $h_1, h_2, \ldots, h_t$. Therefore, after unrolling the RNNs the derivatives of the loss or the gradients contains multiple instances of the same weight matrix W^{hh}. Now if the gradients are small or large which are then propagated through the network, they became exponentially small or large due to continuous matrix multiplication. This is now related to vanishing gradients when it is so small, and vice versa for exploding gradients. This in turn results in the slow training process and sometimes infinite epochs to reach minima or stuck at local minima for longer sequences. Moreover, after unrolling the network for BPTT, this is the problem is more prone to happen.

The exploding gradient problem can be solved by gradient clipping [28], in which a pre-defined threshold is set and the norm of the gradients that surpasses the threshold value is clipped. This prevents gradients to become too large. On the contrary, vanishing gradients are more challenging. To solve this problem, new neural network architectures are developed. LSTM is the most suitable in the family of RNNs, which is discussed next.

2.3.2 Long Short-Term Memory Networks

LSTMs were first proposed by Hochreiter and Schmidhuber in 1997 [29]. The algorithm is designed to learn the long-term representations in the sequential data, which ultimately solves the vanishing gradient problem discussed in the above section of RNN. Unlike RNN, the multiple small units are introduced to process input in a better way and linked together to store the long-term relevant information in a chain-like architecture like RNNs. LSTM units or a memory cell are comprised of an input gate, cell state, forget gate, and an output gate. The cell state has a major role to store the information and gates control the new or old information to keep stored in the memory of the LSTM, i.e., cell state.

2.3.2.1 LSTM Architecture

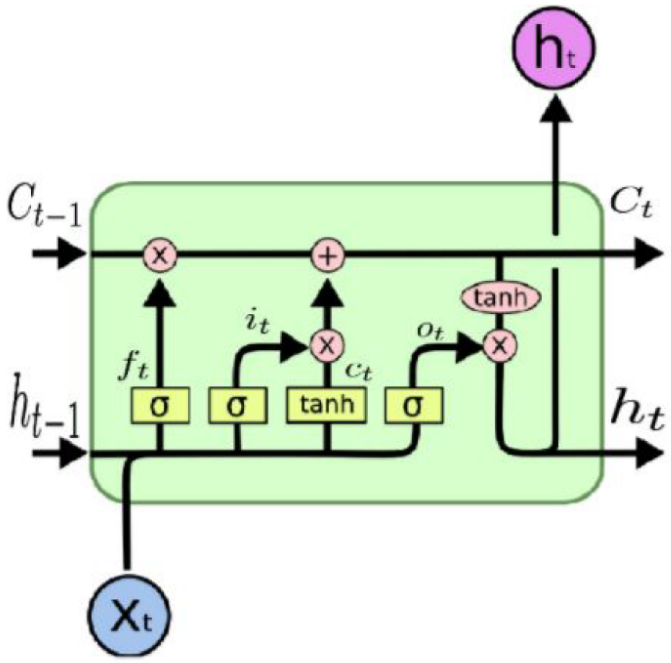

Figure 2.3: LSTM cell Architecture
[30]

As shown in the Figure 2.3, the LSTM unit is composed of a cell, an input, an output, and a forget gate. The cell state C_t represents the memory of the unit with weight 1 allows little linear modifications from the previous cell state. This

state ensures the state of the information as the learning process moves from one-time step to the next. LSTM consists of three gates to control the cell state [30]. Gates comprise activation function and point-wise multiplication. Sigmoid activation outputs the value between 0 and 1 and controls the amount of information to be let through. The mathematical formulations which are used to compute the values over states are as follows:

- **Forget Gate:** When the memory is no longer significant and old enough to remove, LSTM decides to reset using a sigmoid layer called forget gate. This is possible by processing new values in the sequence. Forget gate takes x_t and h_{t-1} and sigmoid activation function is applied. The result f_t is then multiplied by the cell state from the previous time step C_{t-1} and allowing the memory content that has more relevance in the context, and the rest of the information is discarded. This is possible with outputs 0 and 1. 0 means to be discarded and 1 means to keep. This formula is as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.21}$$

- **Input Gate:** This gate reads input x_t and h_{t-1} and is applied to the sigmoid activation function after computing a weighted sum.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.22}$$

- **LSTM Input:** A similar process happens here as in the input gate, except *tanh* activation function is used to produce c_t.

$$c_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.23}$$

- **Memory Cell:** The cell state is updated with all the values calculated above. This includes a Constant Error Carousel (CEC) that has a recurrent unit weight edge [25]. The C_t is calculated by removing irrelevant information from the current input. The next equation describes the computation.

$$C_t = f_t * [C_{t-1}] + i_t * c_t \tag{2.24}$$

- **Output Gate:** Again, the weighted sum of input x_t and hidden state of last time-step h_{t-1} is used by the output gate to diagnose what information goes out of the LSTM block by applying the sigmoid activation function as shown below.

$$o_t = sigmoid(W_o[h_{t-1}, x_t] + b_o) \tag{2.25}$$

- **Output:** Applying the *tanh* activation on the cell state C_t and multiplying with the value from output gate o_t, h_t is calculated, which acts as an output of the LSTM unit for the processing of the next input value in the sequence.

$$h_t = o_t * tanh(C_t) \tag{2.26}$$

In this research, LSTMs are used with different layers and hyperparameters. Though the LSTM is a very popular version of the RNN family and widely used structure, it processes the inputs sequentially to keep the sequence information in the input. Due to that, the training process becomes slow when sequence length is large and/or network size is large. The one more drawback is, due to multiple gates, the computation is performed at multiple places which causes an increase in the number of parameters. The large networks can perform better with respect to time and accuracy, asynchronous stochastic gradient descent (ASGD) optimization process can be applied [31]. The new state-of-the-art model called Transformers solves this problem of processing sequence sequentially. It processes full sequence parallelly by providing position encoding of inputs and results in faster computation. It is intriguing to see this type of neural network work with multivariate time-series data.

2.3.2.2 Regularization Techniques

The following methods are used to improve the performance of the LSTM neural network here in the experiments.

- **Dropout:** As the network size increases, combining several hidden layers and hidden units in the hidden space, the training process of the neural network becomes slower and prone to overfitting. Using the parameter dropout reduces the overfitting in the training process. It randomly drops some of the neurons from the layers and prevents the co-adaptation during the training. The generalized features are learned in the training process by reducing the dependence on some neurons.

- **Early stopping:** In the training process of large networks, the loss decreases gradually but in the case of validation loss, suddenly it starts increasing and ends with a better validation error and results in the worst test error. This problem can be tackled by stopping the training process in between when the validation error starts increasing after some thresholds. This mechanism is called early stopping. After some intervals validation accuracies are stored, and when it starts increasing, the training process stops, and the best values are returned.

There are many other regularization techniques mentioned by author Ian Goodfellow in the book Deep Learning [32], which can be applied to the tasks depending on the nature of data and network type.

2.3.2.3 Bidirectional LSTM

As the name describes, LSTM is applied twice over the input sequence. The input sequence is applied in the forward direction to one LSTM and, the same sequence is applied in the backward direction too. The same approach is also possible for other neural networks from the family of RNNs. The need is raised to understand the context of the sequence that is not seen yet, which is highly needed in NLP tasks like machine translation, text summarization, etc. The significance of this approach is, at any given time t, an LSTM unit has information from past as well as future. The memory can work efficiently in this way to keep and discard information from the cell state.

- **Bi-LSTM architecture**:

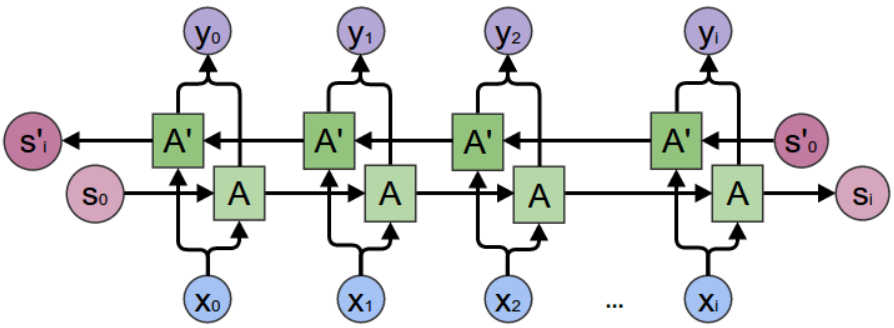

Figure 2.4: Architecture of Bi-LSTM
[33]

The figure represents two LSTM chain-structured in opposite directions. A and A' are the LSTM units who are functioning as described in the previous section 2.3.2. As it is shown in the Figure 2.5, the input sequence $x_0, x_1, x_2 \ldots, x_i$ is applied to both the LSTM units. S_0 to S_i and S'_0 to S'_i are hidden outputs of LSTM unit which forwarded for the next unit's processing. The difference between single and Bi-LSTM is the output calculation. The $y_0, y_1, y_2 \ldots, y_i$ are the outputs used for the prediction or classification tasks in the next layer. The outputs are the average values generated by both directions of hidden state values.

In this research, an experiment with Bi-LSTM network is also performed to witness any improvement over single LSTMs. Though the training time required here is relatively high and has large memory due to larger learning parameters.

2.3.3 Gated Recurrent Unit

Another variant of RNN is Gated Recurrent Unit (GRU) introduced by KyungHyun Cho, et al. in 2014 [34]. This neural network is also proposed to solve the vanishing gradient problem in RNNs. It can be called a variation or an improved version of LSTM, because, in both networks, gates are introduced to control the information available in the memory by reducing the number of learning parameters. GRU does not have a separate cell state to keep the information. The hidden state works as memory and sends information to the next unit. GRU has fewer parameters than LSTM, so it is faster to learn for sequences.

2.3.3.1 GRU Architecture

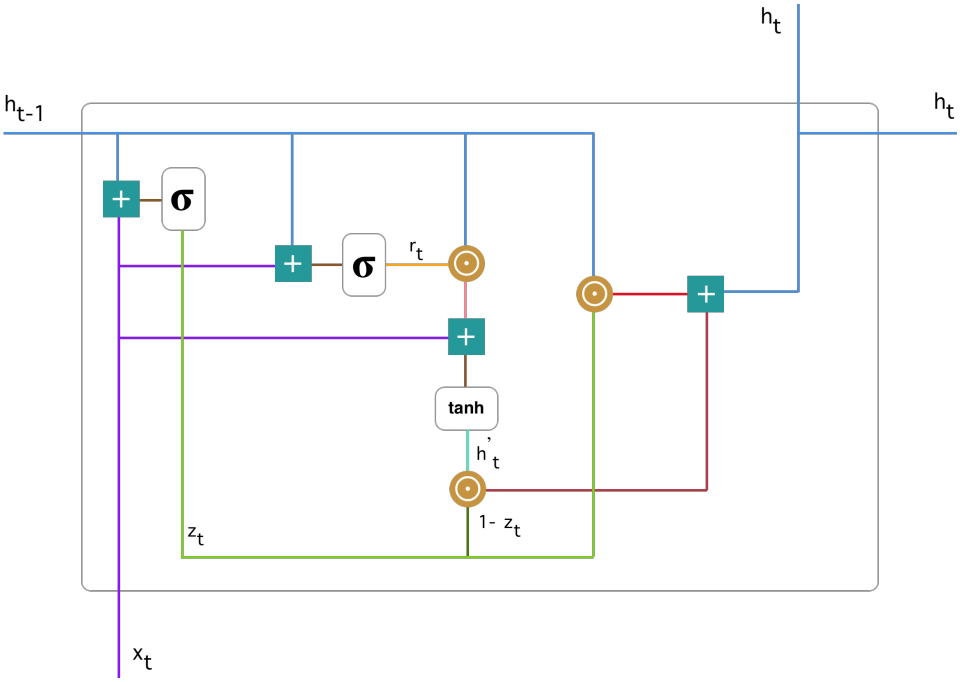

Figure 2.5: Single unit of Gated Recurrent Unit [35]

As depicted in the figure, GRU also comprises gates to regulate the information as it is there in LSTM, update gate and reset gate. These gates are trained to keep the long-time information that is necessary for the correct prediction, and unnecessary information can be discarded. The two activation functions are used to maintain the information in the memory, *sigmoid* and *tanh* functions act processing functions. The mathematical expression at the gates is described further:

- **Update Gate:** The input x_t and hidden state of the last time $t-1$ are multiplied by its weight matrices and provided for the *sigmoid* activation function after addition. The result is between 0 and 1.

$$z_t = \sigma(W^z x_t + U^z h_{t-1}) \tag{2.27}$$

Update gate exactly works as forget and input gate from LSTM. It helps the unit to find how much information from the last time-step is retained and utilized in the future. Because of information retention, the risk of getting training process in vanishing gradient is suppressed.

- **Reset Gate:** This gate mainly focuses on the information to forget from the hidden state.

$$r_t = \sigma(W^r x_t + U^r h_{t-1}) \tag{2.28}$$

The mathematical representation seems the same as the update gate, the weight information is different in both input vectors for the reset gate. This gate is utilized differently than the update gate.

- **Memory Content:**

$$h_t{}' = tanh(W x_t + r_t \odot U h_{t-1}) \tag{2.29}$$

With the above equation, the reset gate value r_t gets multiplied with the previously hidden state h_{t-1} which is first multiplied by weight matrix U. Then addition is performed between the first multiplied output and input which is finally applied to the nonlinear *tanh* activation function.

- **Output:** In the last step, the GRU unit calculate the hidden output vector for the next unit after the changes for the current unit is done, and this process is described by the mathematical equation:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t{}' \tag{2.30}$$

First, the update gate and hidden state from the last time steps are point-wise multiplied. Similarly, the multiplication is applied to $1 - z_t$ and the memory content h_t. Then at the end both results get added to generate the final h_t.

Even if GRU outperforms LSTM in the training speed, the researchers try both of their problem tasks to verify the performance and then choose the best model. In this thesis, to verify the performance of multivariate time-series motions, both the models are used and analyzed.

2.4 Transfer Learning

Transfer learning is defined as the knowledge acquired by the learning process of one task and the dataset is applied to another task and dataset having similar data types and formats to some extent. Throughout this report, the source dataset is referred to the industry dataset which is used for the pre-training process and the target dataset is the phume dataset used for fine-tuning the network after transferring the pre-trained network.

Sometimes transfer learning is compared and confused with the domain adaptation process [36,37]. In this process, source and target datasets are used for training to reduce the dissimilarities between both datasets. One of the applications presented on this approach is to detect the number of people in an indoor environment based on the carbon dioxide percentage in the room [38]. In [39], the human activities are determined using hidden Markov models' generative properties in this domain adaptation approach. Moreover, the authors designed a CNN-based attention model for the encoding of time-series in a supervised way [40]. Pre-training is performed jointly on multiple datasets with domains [41], and it is limited to only one that matches the domain from the target dataset.

When neural networks are initialized, the values generated in weight and bias matrices are random. Therefore, a large dataset is required to understand the data patterns with proper hyperparameters, especially for large networks where a large set of network parameters exists. Due to transfer learning, the initialization process becomes easy such that some knowledge/pattern is already present, which then helps in better prediction. Transfer learning is another form of optimization that speeds up the progress and performance while modeling the second task, provided the enormous resources to train the deep learning models. The transfer learning makes the most sense for task A transferring to task B when both tasks have the same input x, the data for task A is more than task B and low-level features from task A could be helpful for learning task B. Generally, a higher start should be required with the source model in the transfer learning process. The similarly higher slope shows the rate of improvement in skill transfer. The convergence factor on the pre-trained model should be high, called a higher asymptote [42]. In the reverse case, where the data in task A is less than task B, then performance might not be improved in most of the cases.

There are many pre-trained models are available by large research organizations where models are trained over vast datasets, such as a corpus with millions of entries, extensively for many days or weeks and often release their final model for reuse under a permissive license. An organization called Hugging Face [9] provides many pre-trained models, and other organizations can use those models under certain reusable licenses.

Chapter 3

Experimental Setup

This chapter comprises the type, format, labels involved of data, how the datasets are prepared and used in the training process, implementation of the neural networks, and set up the process to execute the experiments, i.e., creation of a pipeline.

3.1 Dataset

Two datasets are used to perform the experiments. Phume dataset is prepared under the observation of Honda Research Institute. The other dataset is the industry dataset [43] which is recorded for the improvement of industry conditions and activities using collaborative robots. Data is similarly collected at HRI by setting up the popular Xsens bodysuit recording 23 different body joints [44] with motion sensors measuring velocity and accelerations parameters with linear, angular, etc. motions for two hours. The combination of each body joint and sensor motions along with three-dimensional coordinates points results in many columns for a single time t for the Phume project. The total sensor points are 329201 and 1740 columns for the phume dataset and the industry dataset has 1510683 points with 858 columns.

The sample rate used here while recording motions is 60 Hz, which is normally enough to track dynamic tasks. It means to represent a single second, there are 60 values for every column. A few columns are subject_id, grouped_sequence_id, sequence_id, and label. Rest all columns are related to motions with coordinates. Every person involved in the experiment to record the data has assigned a random number to keep the anonymity. The sequence_id is the number assigned to an activity, i.e., if the walking is an activity and data is recorded for a minute then for 60 seconds the 3600 sensor points are recorded as the device is measuring at 60 Hz.

In the industry dataset, the Label column represents the motion action for that time. There are many categories for that, having multiple columns. There are General posture labels with 4 categories as Standing, Walking, Crouching, Kneeling. The other label column is a detailed posture with 12 categories like standing upright, standing bent forward, etc. So the general posture is further categorized in detailed labels. The third label category is current actions with 8 categories like idle, react, pick, carry, etc. All detailed information is available in the paper [43]. There is only

one label column in the phume dataset which has 20 categories in total, such as stand, walk backward, turn around, etc. which are also available in brief in the paper [13].

3.2 Implementation

Both the datasets have many columns, and it is not possible to use all columns due to the curse of dimensionality [45]. Rather it is not required to use all columns, few columns are sufficient to learn the pattern in the data that predicts the correct label. So, for the transfer learning process, the industry dataset is used for pre-training the network and using the weights for the second task with the phume dataset. In this thesis, for the experiments, position and sensor acceleration are the two types of columns used. Sensor acceleration is the raw motion data points recorded by the bodysuits and the position columns are the points prepared by considering other points like the value can be calculated from sensor acceleration, sensor velocity, angular velocity and acceleration, and many more, which depicts the point of the motion in the time t.

Index	Segments	Description	Origin of the segment frame
1	Pelvis	Segment between both hip joints and joint L5S1	Midpoint between right and left hip center of rotation
2	L5	Segment between joints jL5S1 and jL4L3	jL5S1
3	L3	Segment between joints jL4L3 and jL1T12	jL4L3
4	T12	Segment between joints jL1T12 and jT9T8 and jL4L3	jL1T12
5	Sternum (T8)	Segment between joints jT9T8 and jT1C7	jT9T8
6	Neck	Segment between joints jT1C7 and jC1Head	jT1C7
7	Head	End segment above joint jC1Head.	jC1Head
8	Right Shoulder	Segment between joints jRightC7Shoulder and jRightUpperArm GH	jRightC7Shoulder
9	Right Upper Arm	Segment between joints jRightUpperArm GH and jRightElbow	jRightUpperArm
10	Right Fore Arm	Segment between joints jRightElbow and jRightWrist	jRightElbow

Index	Segments	Description	Origin of the segment frame
11	Right Hand	End segment after joint jRightWrist.	jRightWrist
12	Left Shoulder	Segment between joints jLeftC7Shoulder and jLeftUpperArm GH	jLeftC7Shoulder
13	Left Upper Arm	Segment between joints jLeftUpperArm GH and jLeftElbow	jLeftUpperArm GH
14	Left Fore Arm	Segment between joints jLeftElbow and jLeftWrist	jLeftElbow
15	Left Hand	End segment after joint jLeftWrist.	jLeftWrist
16	Right Upper Leg	Segment between joints jRightHip and jRightKnee	jRightHip
17	Right Lower Leg	Segment between joints jRightKnee and jRightAnkle	jRightKnee
18	Right Foot	Segment between joints jRightAnkle and jRightToe	jRightAnkle
19	Right Toe	End segment after joint jRightToe.	jRightToe
20	Left Upper Leg	Segment between joints jLeftHip and jLeftKnee	jLeftHip
21	Left Lower Leg	Segment between joints jLeftKnee and jLeftAnkle	jLeftKnee
22	Left Foot	Segment between joints jLeftAnkle and jLeftToe	jLeftAnkle
23	Left Toe	End segment after joint jLeftToe.	jLeftToe

[44]

Table 3.1: Description of segments / body joints

There is a total of 23 joints of the body which is measured by the bodysuit [44] and considering three coordinate systems, it becomes 69 columns for a single motion. Again, it is so much data to train with limited resources. So, 11 body joints are selected for position column type which are the pelvis, l3, sternum (T8), neck, head, left and right shoulder, left and right forearm, left and right lower leg. These segments are selected to cover the full body joints. The joint names are associated with index numbers in Table 3.1 and the same numbers can be seen in the bodysuit avatar Figure 3.1 to understand the body joint in detail. With the Cartesian coordinate system, it is 33 columns. The same joints are considered for the sensor acceleration

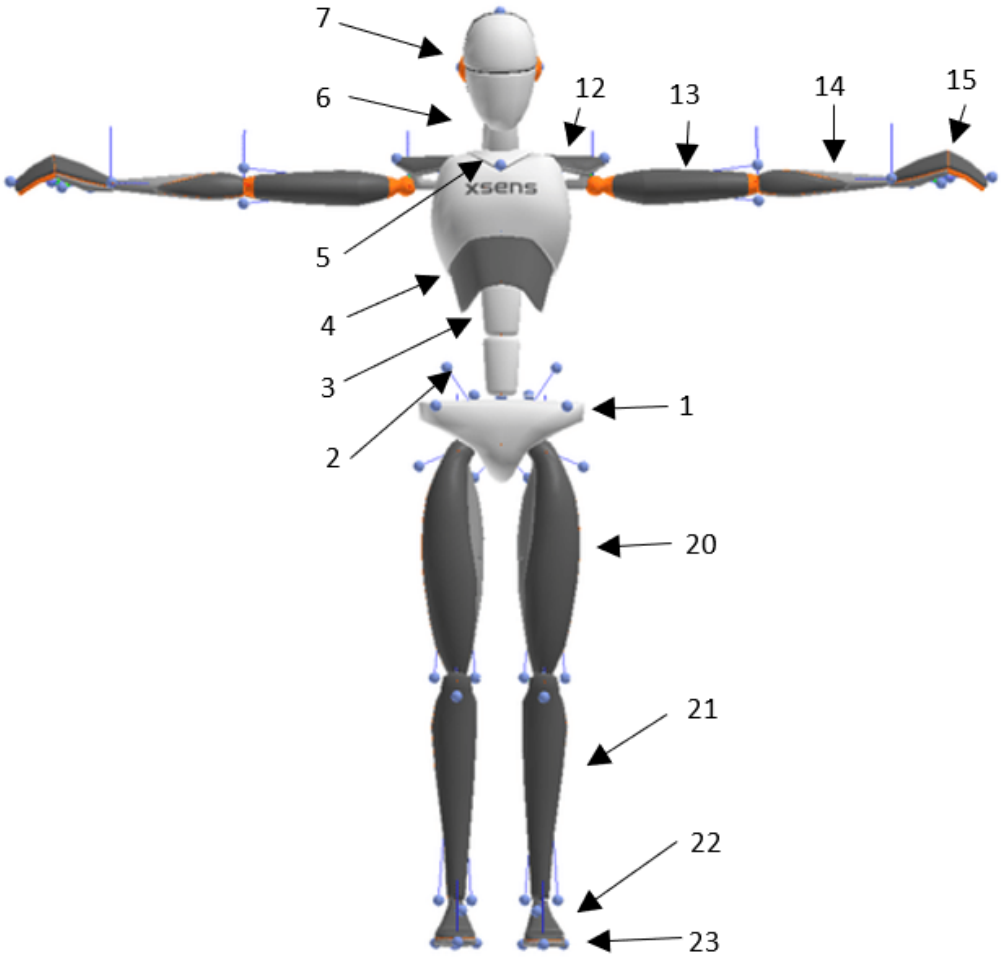

Figure 3.1: Indication of the different segments in the avatar [44]

column type except, l3 and neck. From the industry dataset, it has been decided to use labels from detailed posture which are 12 in total and 20 labels in the phume dataset are very close enough and related motion, so the neural network could learn better in hidden layers. Still, this argument does not hold strongly true because, in the transfer learning process, the last classifier is going to be replaced.

The data points from the datasets cannot be selected randomly for the training process. The data point has a sequence number, and this sequence number represents an action. One of the major properties of time-series data is its sequence, so for every repetition, the sequence IDs are selected randomly at 70% for the training process, 10% for validation, and 20% for testing. Now, the main algorithm of experiments is to take data points from the sequence IDs and convert them into a window frame. The window size is 60 because the data is recorded at the rate of 60 Hz. It is possible to take smaller or larger window sizes. With larger window sizes there are always be problems with the availability of hardware resulting in the slow training process. Data points are arranged in this window size in such a way that for a particular sequence ID, all the first 60 points are considered to be one-time

frames and the label for the 60th row is attached for that frame. Then the window is a slide by 1 position to next and then from that point to next 60th points are the next time frame. This means this is an increase of the data in total for the training process.

This process can be analyzed by the following example for the calculation. Let's say there are 4000 sequence IDs. Random 70% of these 4000 sequence IDs is 2800. 400 IDs are for validation and 800 IDs are for testing. Consider there are 400000 data points, and each sequence ID holds 100 data points, which is not true for the datasets. It varies a lot, but for simple calculations, the assumptions are made. The 280000 x 33 (columns) data points are used for the training process. As stated, 60 points are considered for the time frame and slide by 1 which results in 279940 times a matrix of 60 x 33 data points, and the same number of labels are associated with the matrices. All the experiments are repeated 10 times to avoid bias and randomness for a particular seed point. Moreover, the prepared matrices are used in the portions to find the performance of the network in increasing order of data usage. In the chapter 4 further, it is seen that from 10% to 100% data is used for both the training and fine-tuning process of the phume dataset. But all prepared data points are used for the pre-training process of the industry dataset. To explain with the example mentioned above, the first 10% of 279940 is 27994, matrices are used at the start and then the first 20% of 279940 is 55988, matrices are used of 2nd time which also includes data from the first step.

3.2.1 Usage of PyTorch

The neural networks are implemented in Python programming language using PyTorch and PyTorch Lightning libraries. PyTorch is the library built by Facebook's AI research lab and provides so many built-in libraries to make research in the field of Deep Learning. The way of writing code in PyTorch is very easy, and majorly it is with object-oriented and functional paradigms. Similarly, the PyTorch Lightning provides a high-level interface for PyTorch and makes doing experiments very fast. It takes care of all engineering processes by wrapping all code execution in methods so-called hooks and calling of methods are completely managed by PyTorch Lightning.

The usage of PyTorch lightning is explained further in brief. The prepared data matrices are used with Pytorch's Dataset and Data Loader module. Since the full data cannot be loaded into memory for the training process, therefore, it is useful to use the data loader with provided batch size. The neural network is prepared by extending the PyTorch's nn.Module in a new class and prepare a custom network, by using PyTorch's LSTM or GRU libraries, followed by a linear layer used classification. The next step and important step is to create a class by extending the LightningModule which executes all the ordered hooks. The best of using PyTorch Lightning is the handling of data in a GPU-enabled system. The data is by default transferred to GPUs or TPUs if available. It also provides a training process to run in distributed GPUs which process the data parallelly and combine at the time of gradients calculation.

Many method parameters are used to tweak the experiments in the training process when the trainer object gets created. Here, only the important parameters that helped in the training process are mentioned. The callback is a very important parameter in which multiple callbacks can be mentioned. The `checkpoint callback` which stores the weight at mentioned storage path after completing the training process have been used here and `early stopping callback`. This early stopping concept is very useful when then there is no increase in the training process. This checks the metric validation accuracy (`val_accuracy`) which is logged after each epoch and checked with parameter patience 30 times. The value for this `patience` parameter is configurable. This means the training process at least runs for 30 epochs and if there is no increase in the validation accuracy then the training process is stopped. `auto_lr_find` is again an important parameter that adapts the learning rate hyperparameter after every epoch. The process of adaption is done by the library.

3.2.2 Usage of LSTM Network in Transfer Learning

The Figure 3.2 depicts the process of transfer learning from the industry dataset to phume. The experiments are performed with multiple layers, but here the 5 LSTM layers process is shown. Each 5 layer has 64 hidden units or neurons. Each neuron is an LSTM cell described in the chapter 2. The interesting part of the RNN type of model is it accepts sequences. So, in NLP, a sentence is a sequence of words, and an embedding layer is added to understand the correlation within the words. Similarly, the window size 60 is acted sequence length and the number of columns (for eg. 33 for position type columns) are embedding layers. Each data point is processed as a single word and the weights of the last hidden layer go to the first hidden layers neurons when the word in the sequence is processed, i.e., the second data point in this case. This way the process of applying sequences to time-series data is unique and makes all deep learning networks open to experimenting with time-series data.

The last layer is the classifier layer with 12 label classes for the industry dataset, and the network can be easily transferred to another task just by replacing the last layer as shown in the Figure 3.2. In the learning process, an `Adam` optimizer [46] is used and `cross-entropy` loss [47] is used to calculate the loss of this classification task.

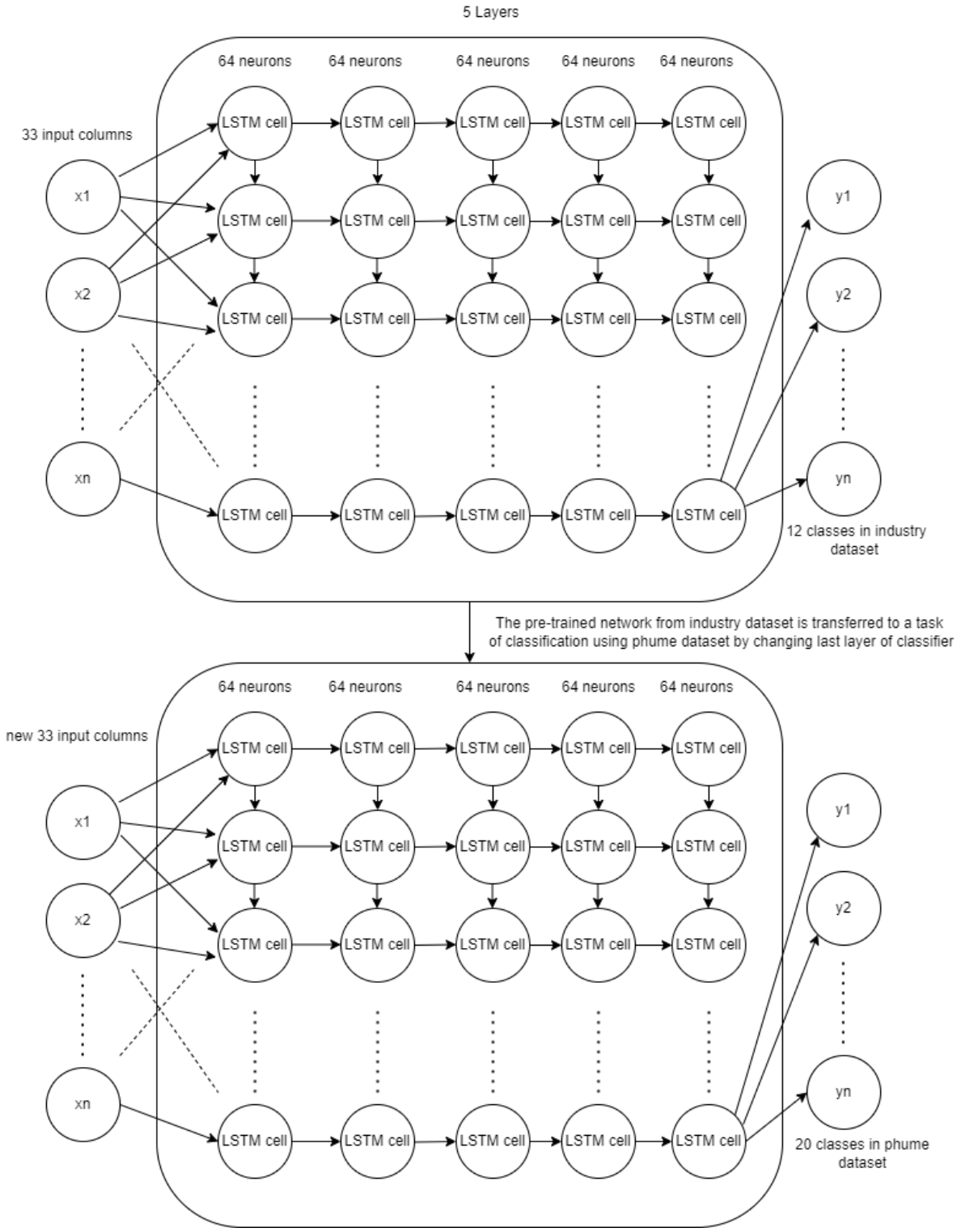

Figure 3.2: A process of transfer learning used in one of the experiments for training with LSTM network of 5 layers by replacing last classifier layer

Chapter 4

Results and Discussion

Considering the networks discussed in the previous chapters, the experiments can be conducted in various combinations. Many parameters need to be set before unleashing the power of pre-training precisely. Some hyperparameters are set to some constant values which are as follows:

Hyperparameters	**Values**
Hiden units / neurons	64
Learning rate	0.005
Dropout	50%
Window size for TS frame	60
Batch size (Phume)	128
Batch size (Industry)	2048
Epochs (min)	30
Epochs (max)	100
Data distribution Train Validation Test	 70% 10% 20%

Table 4.1: Hyperparameters

To measure the effect of pre-training, the accuracies are measured by training the industry dataset and using the checkpoint (representation of the task in the form of weights) for the next task of the Phume project. To find the preciseness of the training process, the training of the phume dataset was performed 10 times in most of the cases to minimize the variance, a standard process to minimize or generalize the randomness in the training. Training process is generally time consuming, especially when the dataset is large and network size is dense. Different networks with varying layers are trained, which took time from almost 30 to 72 hours for all 10 repetitions. The common performance measure used here is the test accuracy. It is the metric measured on test data after the training process. This data is 20% of the dataset mentioned earlier and is completely new and the network has not seen earlier.

4.1 Comparing Networks and Layers

Experiments are conducted with two deep neural networks, LSTM and GRU. It is very interesting to compare the results with various combinations.

4.1.1 For Position Columns

As discussed in the chapter 3 that the position data is calculated from the raw sensor values. So the comparison between the layers distinguish the performance of the pre-training for the position columns.

- **2 layers**

Figure 4.1: Learning process for LSTM and GRU networks with 2 layers each for position columns

In the graphs in Figure 4.1, the red curves are training performed on phume dataset using LSTM and GRU networks with 2 layers with 64 hidden units in the hidden layers. On the x-axis it is the portion of phume data used for the training process. It means 0.1 is the 10% of phume dataset used and so on. On the y-axis, it is the test accuracy recorded on test data which is 20% of the total phume dataset. So in total, data is split into 70-10-20 to train, validation and test respectively, 0.1 means 10% of 70% of total data. It means for the initial few portions, the training data is less than the test data. When it comes to 20%, the initial 20% of 70% is considered, i.e., 20% also consist of 10% data used for first training process. And this is how the overall dataset has been used for training purpose to get more insight when using portions of dataset.

In the graphs, blue curves are the fine-tuning process, reused the pretrained weights representation from industry dataset on phume dataset. The solid lines

are the mean(μ) values of all 10 repetitions and the shaded region represents the standard deviation(σ), i.e., $\mu \pm \sigma$. Some points can be easily interpreted as, on the initial portion of phume data, where data available for training is very less compared to testing, normal training is not able to find the good results in terms of accuracy. Whereas due to pretrained data, the training process in the blue curve can easily classify the task labels. Since this is only 2 layer architecture, the significant difference is hard to identify. For LSTM architecture, a little advantage can be seen from 20% to 60% and from 70% there is no significant difference. There is not a big difference at the initial portions between means in the case of GRU architecture. Considering that test accuracy is also increased gradually as the data used for training is increased.

To provide the results statistically significant, the mean lines should be as far as possible and the shaded region should not overlap. In this case, LSTM seems to be better than GRU because the overall accuracy is better at all stages of the training and the initial 10% portion of training data.

- **4 layers**

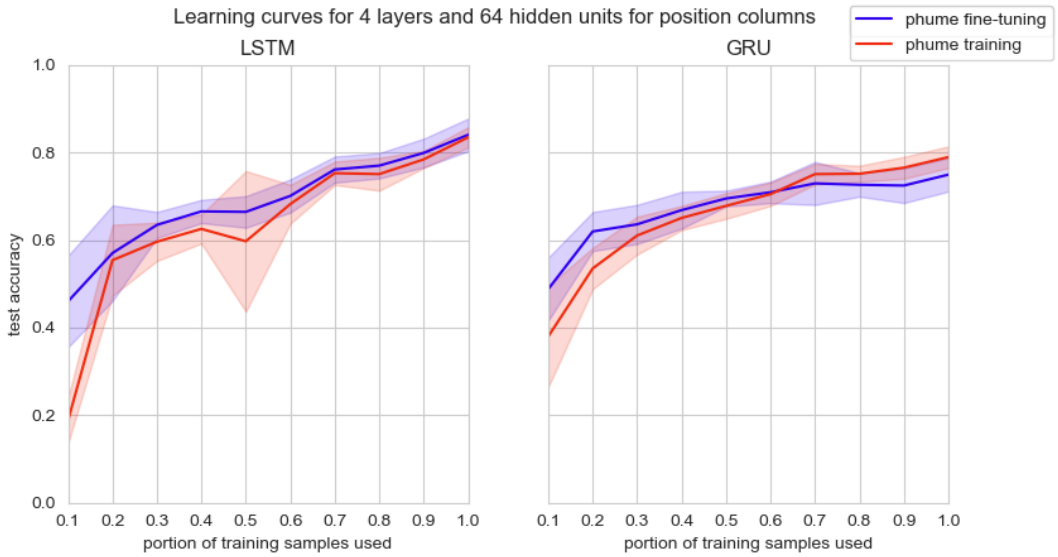

Figure 4.2: Learning process for LSTM and GRU networks with 4 layers each for position columns

For the graphs in Figure 4.2, the results are mostly the same as for 2 layers' architectures with little more deviation for the LSTM case. The initial gain or advantage can easily be seen for LSTM fine-tuning. When all 70% of train data is used for training, the LSTM gives better results compared to GRU. Initially the problem is, the network can not learn enough to classify things precisely with less data.

- **5 layers**

A significant effect can be seen from this architecture in Figure 4.3. When using full data for training, the optimization process fails to find the global minima in the gradient descent process, and accuracy is badly impacted in the case of LSTM. And the pre-training helped here in a very positive way. Performing fine-tuning on the phume data, the results are well reached near to 80% test accuracy. On the other hand, in the case of GRU, the training process did not fail but lagged to achieve better accuracy than LSTM. Pre-training does not provide any significant advantage too except for the initial 10% training samples.

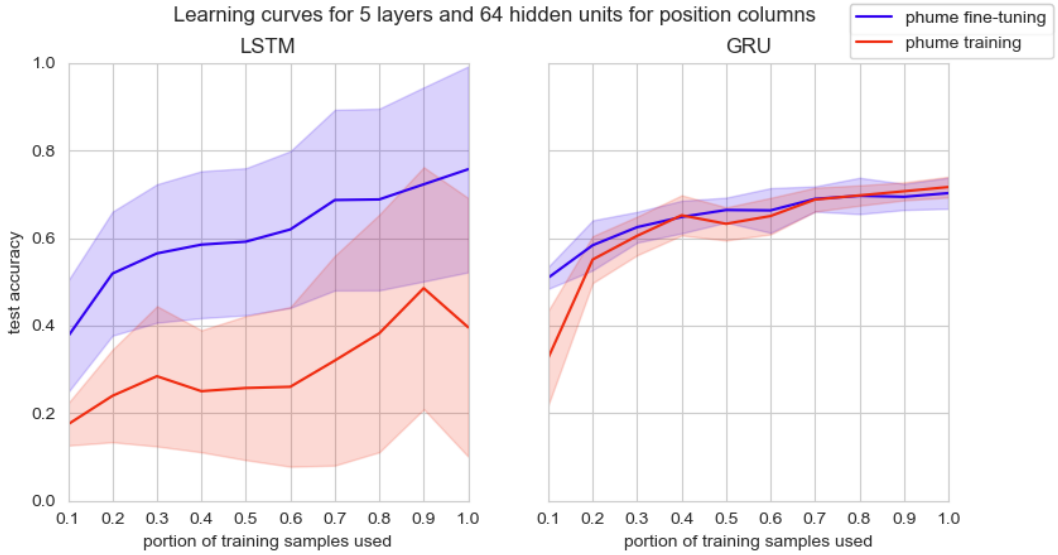

Figure 4.3: Learning process for LSTM and GRU networks with 5 layers each for position columns

In the case of LSTM, the standard deviation is quite large. The results are averaged of 10 repetitions. For one of the repetitions, the training result for the industry dataset is poor and because of that at the time of fine-tuning phume data, the result is badly impacted. The best result out of those 10 repetitions for the industry dataset is 0.9251772165298462 and the worst result is 0.35985255241394043. The same results were tried to reproduce by providing the same seed values, but the results are all good, and, unfortunately, could not find the reason for that badly impacted iteration.

To analyze this case deeply, 40 more times this experiment is repeated, i.e., at the end the mean and standard deviation values are of in total 50 repetitions shown in Figure 4.4 which minimized the effect, and it can be ignored by considering as outlier. Compared to the graph of LSTM in Figure 4.3, here the standard deviation is reduced, and the test accuracy is nearly the same. The normal training process has similar effect.

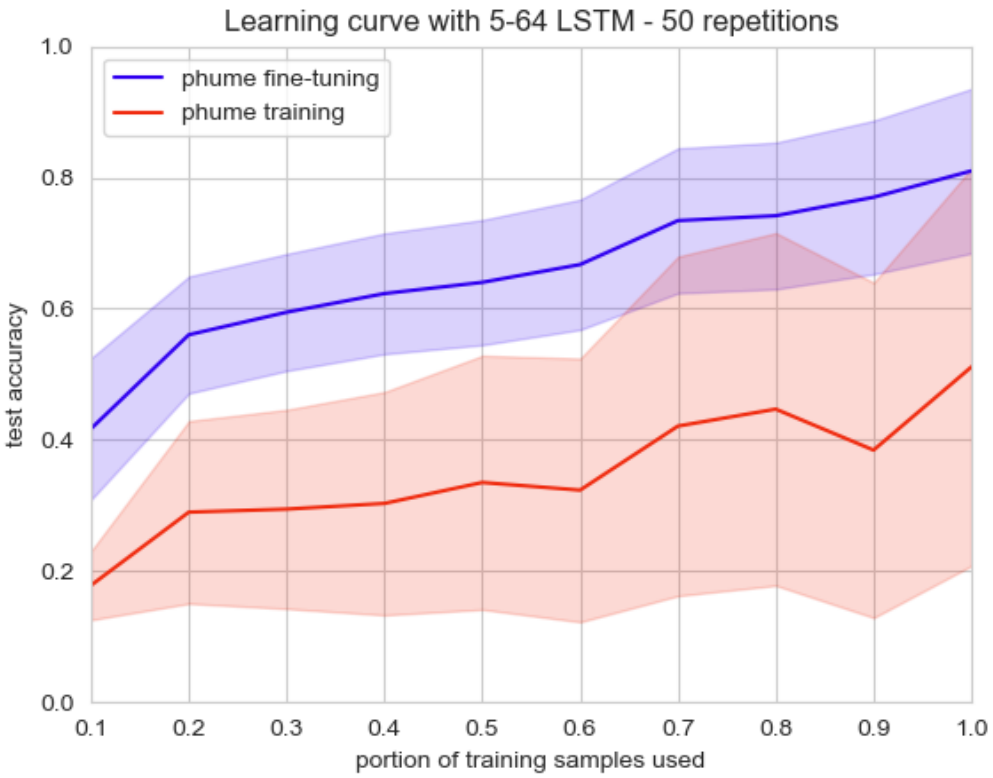

Figure 4.4: Learning process for LSTM network with 5 layers and 50 repetitions

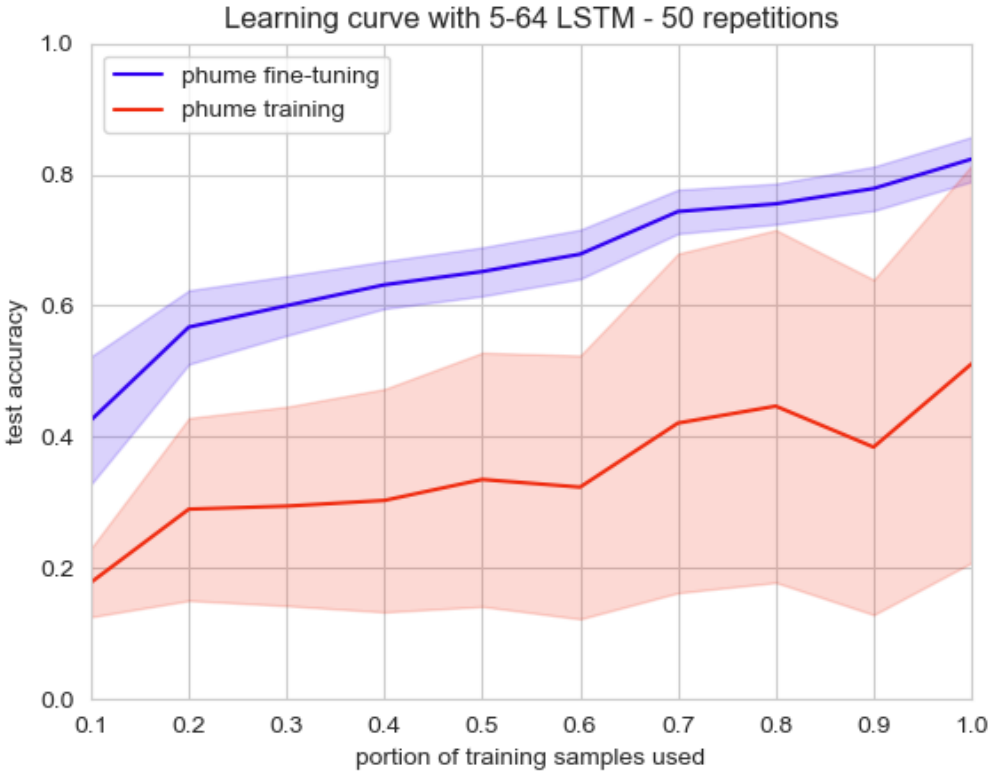

Figure 4.5: Learning process for LSTM network with 5 layers and 50 repetitions (first 10 repetitions performed again with the same seeds used in Figure 4.3)

As mentioned earlier, when the first 10 repetitions performed again with same seed values, the results are good. Because of that, the standard deviation is very less in the new plot of Figure 4.5. This graph clearly shows the benefits of pre-training process here. The result by which the result, is badly impacted, can now be treated as outlier and ignored.

- **6 layers**

Figure 4.6: Learning process for LSTM and GRU networks with 6 layers each for position columns

The same significant effect is available for this architecture too. The LSTM case predominantly describes the power of pre-training. The simple training process failed without any improvement, whereas fine-tuning on phume data continued to provide significant results near 80% test accuracy when using full 100% of 70% phume dataset for training. On the contrary, the GRU's optimization does not fail to achieve classify things in a simple training process but overall does not perform well to best test accuracy when compared to LSTM.

- **7 layers**

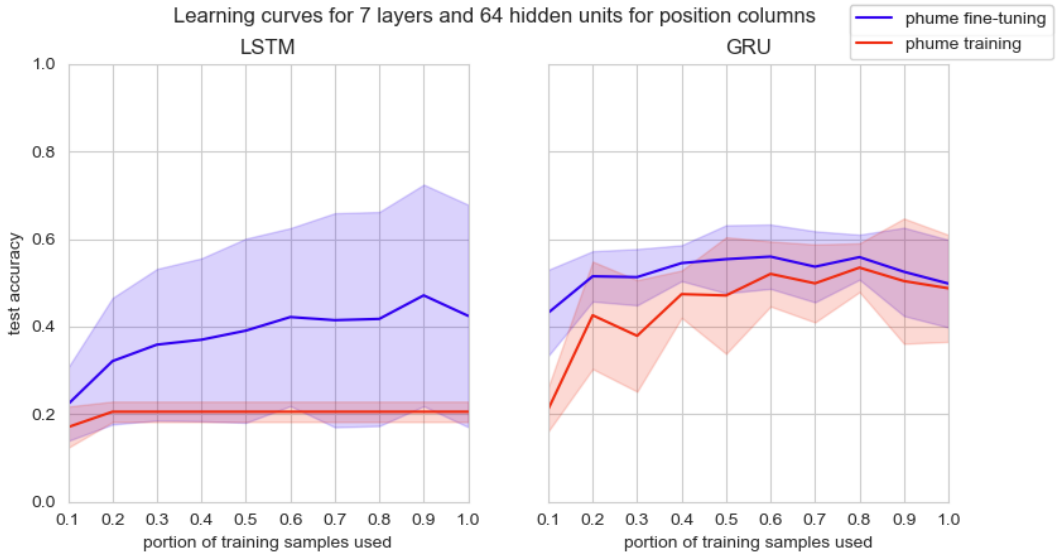

Figure 4.7: Learning process for LSTM and GRU networks with 7 layers each for position columns

From this experiment, LSTM architecture fails for large networks at least for this time-series position columns phume data when used for simple training. To overcome the problem of large networks, pre-training is the best-suited solution in these scenarios. But this time overall accuracy is not good, rather it is less than 50% compared to GRU. In the case of GRU, both training processes are reached near 50%. In both cases, the standard deviations are very high to justify the usage of the networks.

- **8 layers**

For the LSTM architecture in Figure 4.8, the results have no improvement at all and from this structure, LSTMs are not performing well. On the other side, GRUs continues to learn from data but overall, the test accuracy is not good and below 40% with a high standard deviation, which cannot be generalized even with a higher number of repetitions. The reason is very simple that for these much large architectures, the network cannot keep the learned knowledge from the data in memory, in simple terms, it forgets the data seen earlier, due to which the total loss fails to minimize. To achieve better results with large networks, smaller learning rates could be helpful which can take smaller steps in the optimization process.

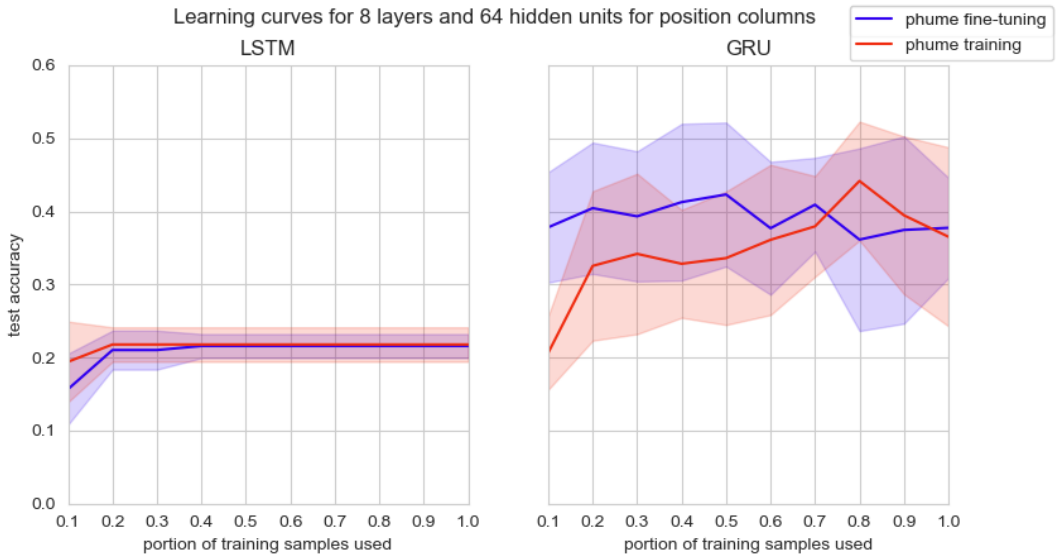

Figure 4.8: Learning process for LSTM and GRU networks with 8 layers each for position columns

4.1.2 For Sensor Acceleration Columns

In this experiment setup, the sensor acceleration columns are considered with different layers of LSTM and GRU architectures. The classification problem using these columns is different from the position columns. Because the sensor acceleration columns are raw values recorded by the IMU and other sensors whereas the position columns are prepared using some math operations performed over other raw columns and finding the position at a particular time.

In all the graphs of Figure 4.9, there is no significant difference (and advantage) between a normal training of phume dataset and fine-tuning it. The comparison between networks is also challenging as there is no such any difference in test accuracy. The same indifferent behavior can be seen for all layers. In the end, the training process gradually improves the performance as the data used for it, is increased. The pre-training on industry data set is not contributing as a point as it is showing significant improvement for positions columns over an increasing number of layers. Only the increased standard deviation can be observed on increasing the layers, something which can be justified that for large networks it needs large dataset or smaller learning rates which would decide the small steps towards global gradient. The main reason could be, that sensor acceleration columns are values recorded directly by sensors, so it is substantially easy to predict the next data point and classify the motion. In contrast, position columns are calculated after processing other columns, in which it is the previous data points does not contribute directly and the network needs to find a pattern to recognize the points in next time stamps.

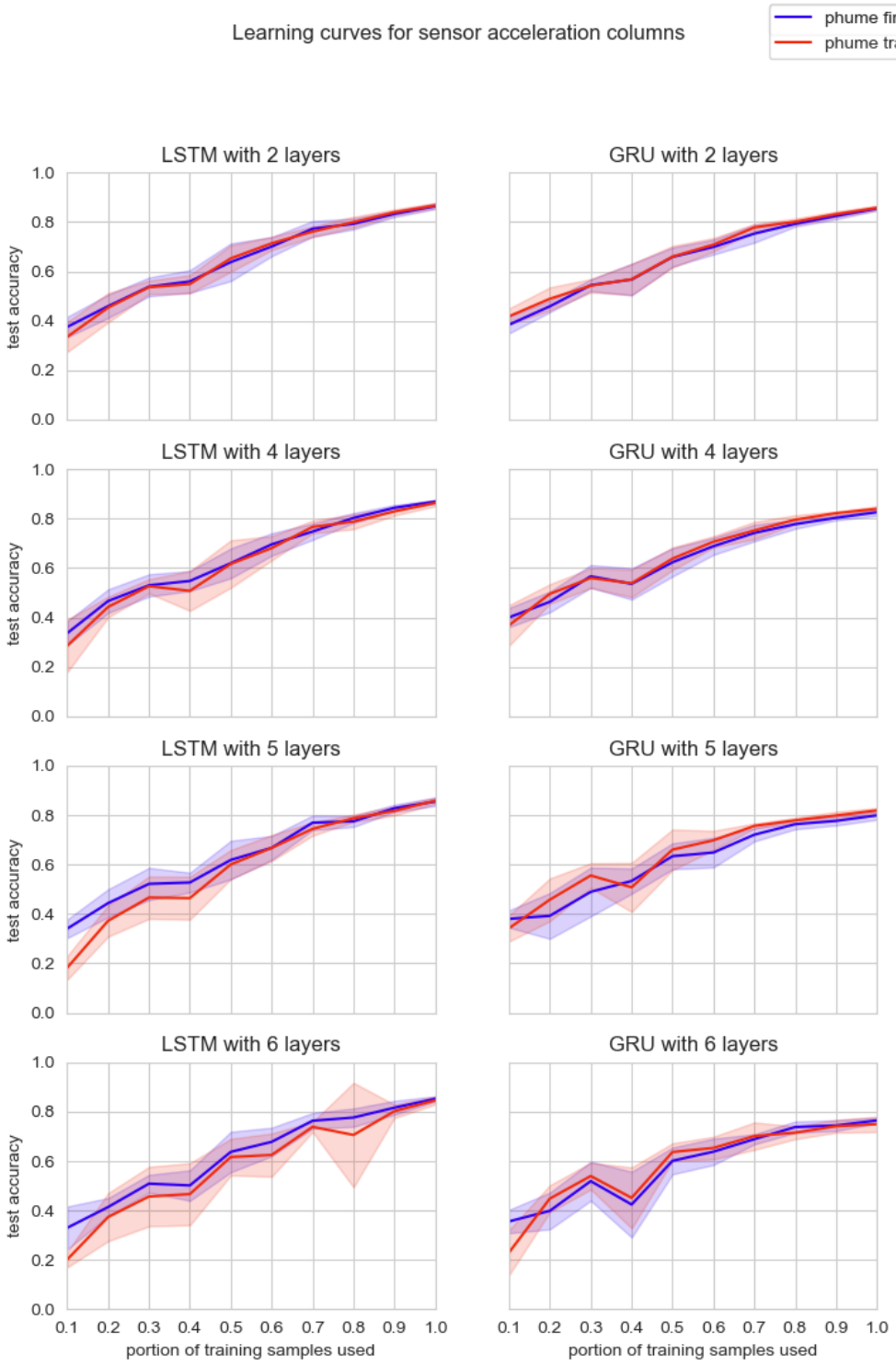

Figure 4.9: Learning process for LSTM and GRU networks for 2 and 4 to 6 layers each for sensor acceleration columns

4.2 Partial use of Industry Dataset for Pre-training

4.2.1 For Position Columns

In this experiment, the training process is done on a dataset consisting of position columns. As the industry dataset is very large labeled data and easily available for the experiment. But this is not the scenario in real life. A huge unlabeled data is generated every second nowadays and to convert it into labeled data is a very tedious and time-consuming task. So usage of industry dataset is limited to small portions initially and increased gradually. In the first experiment 4.1, it is observed that GRU is performing less compared to LSTM in terms of test accuracy, so the experiments from here are limited to LSTMs only.

- **2 layers**

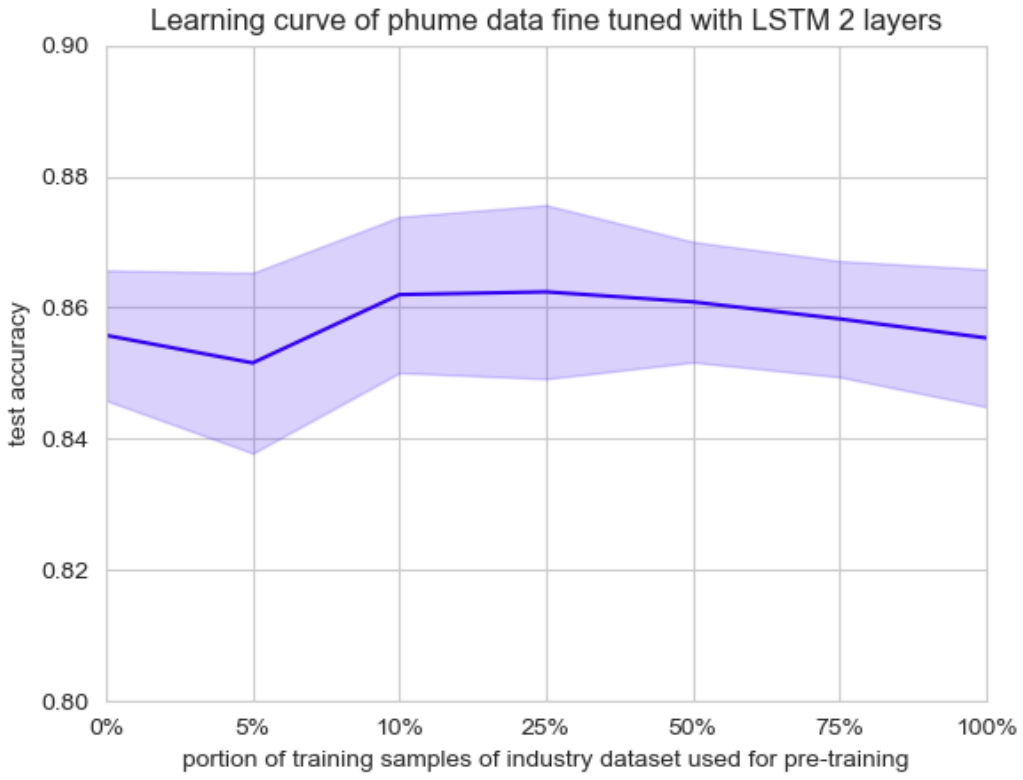

Figure 4.10: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 2 LSTM layers with position columns

In the above graph Figure 4.10, on the x-axis the percentage of industry dataset used for pre-training is shown, i.e., the first value is 0% which indicates there is no pre-training involved. Only phume dataset is used with all points in it, i.e., all 70% is used for the training process. The second value is 5%, which means that much portion of the industry dataset from start is used for training. Then these pre-trained weights are then used to fine-tune the complete phume dataset. The y-axis is the test accuracy calculated after training.

This graph is for 2 layers of the LSTM network, in which it is overall achievement is not that significant. The mean value is near between 85 to 86% and the standard deviation is not varying too. In total, the pre-training does not help much irrespective of the portion on which it is pre-trained.

- **4 layers**

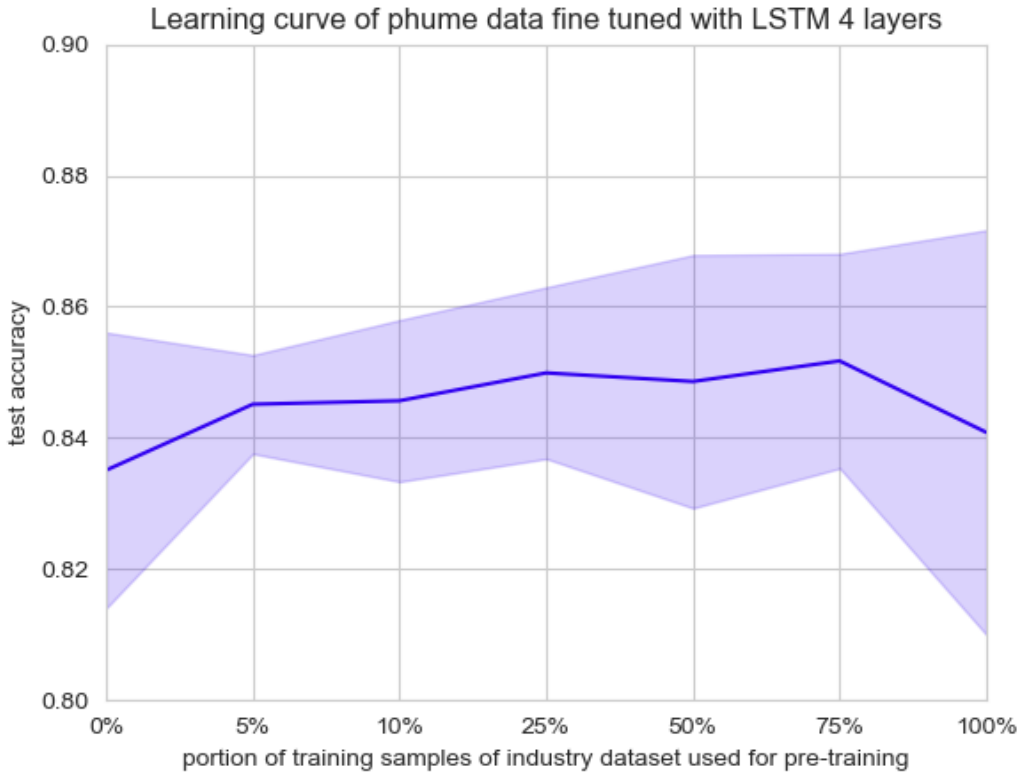

Figure 4.11: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 4 LSTM layers with position columns

As the layers are increased in the LSTM architecture, it shows little improvement in the above graph Figure 4.11. The 5% pre-trained model impacted a little and improved the test accuracy. And it maintained it over the increase of percent usage.

- **5 layers**

The graph in Figure 4.12 shows that there is a significant increase from 45% to about 70% only with 5% of pre-trained data considering high standard deviation. This effect is most noticeable for large networks. It can be narrowed down towards mean with more repetitions. Further, increasing the usage of industry datasets in pre-training, the test accuracy crossed 80%. This can be useful for online learning as well, in which a right assignment of weights can boost the performance for classifying data well.

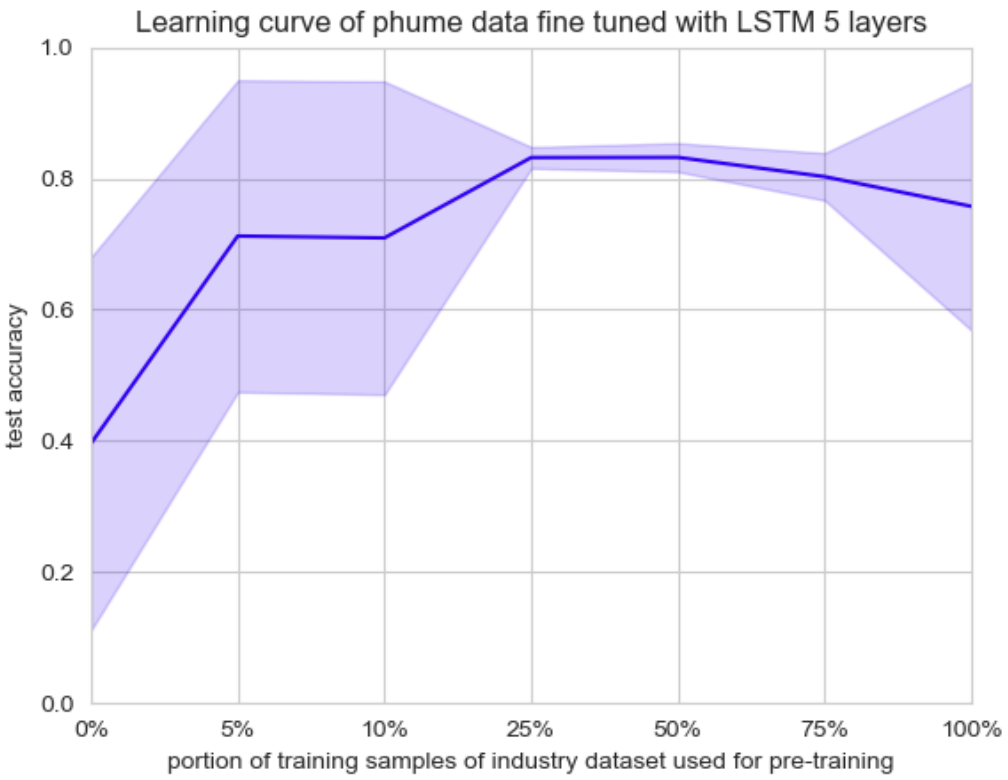

Figure 4.12: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 5 LSTM layers with position columns

- **6 layers**

  For 6 layers, directly training on the phume dataset results in poor performance of the network. With an initial 5 and 10% portion of the industry dataset, pre-training improves slightly more. But with 25% usage of the industry dataset for pre-training, gives noticeable improvement. This again shows that as the network size increases, a large percentage of pre-training is required. This is because initially the weights are initialized randomly and with pre-training the initial identification and finding pattern into transferred task make easy for small datasets.

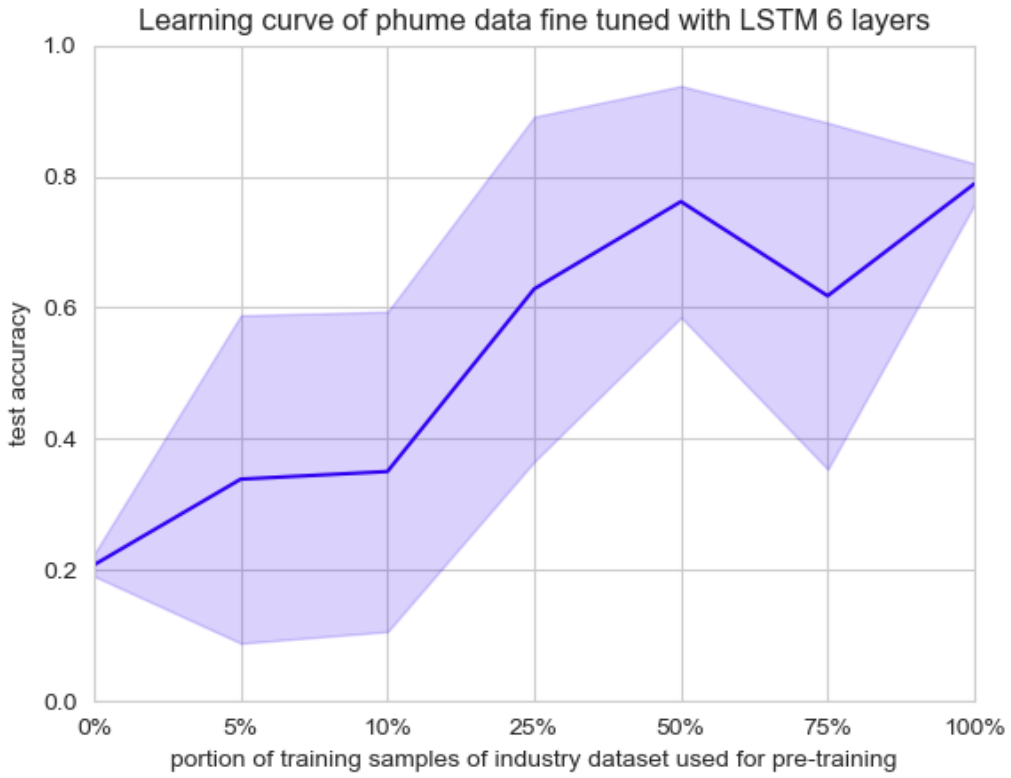

Figure 4.13: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 6 LSTM layers with position columns

- **7 and 8 layers**

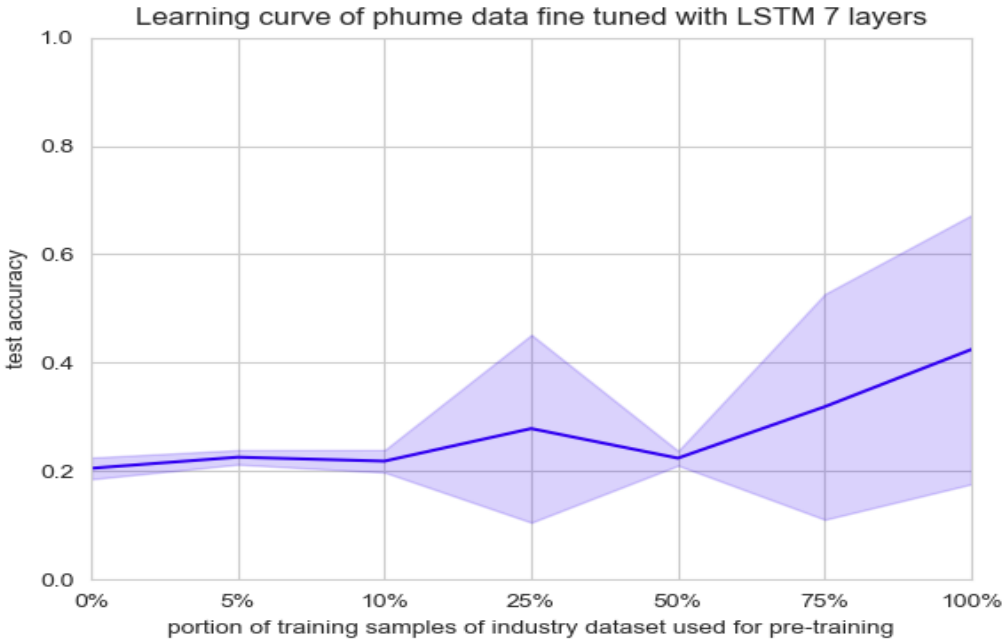

Figure 4.14: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 7 LSTM layers with position columns

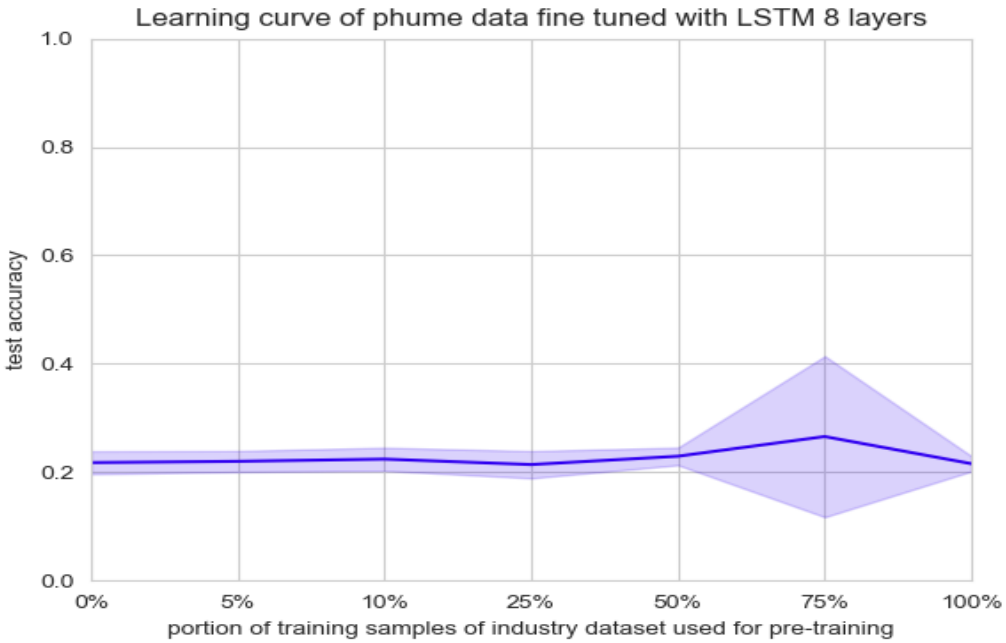

Figure 4.15: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 8 LSTM layers with position columns

In the graphs of Figure 4.14 and Figure 4.15, with 7 and 8 layers the networks become large, because of that the available data is not enough for the training process and classify actions correctly. The optimization is failed. The overall test accuracy is very poor. The standard deviation is high for a few points, but it is not significant.

4.2.2 For Sensor Acceleration Columns

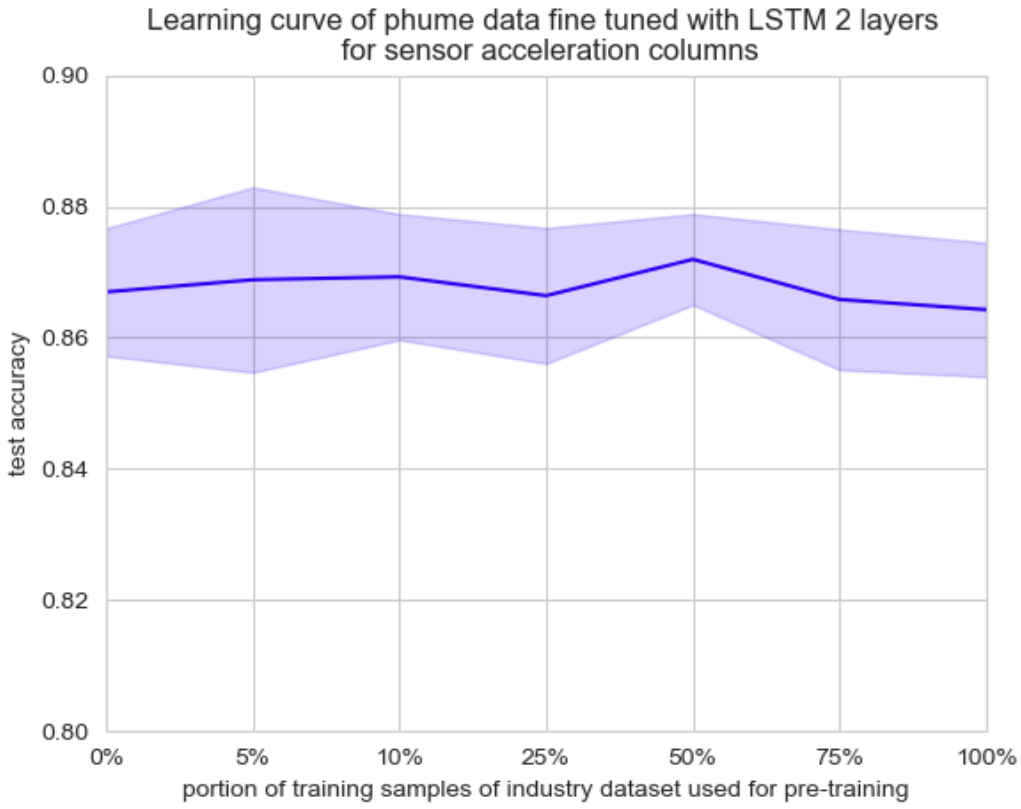

Figure 4.16: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 2 LSTM layers with sensor acceleration columns

Unlike the results from position columns, there is no significant effect in the case of sensor acceleration columns. Even for the higher layers, simple training is contributing the same way as with the pre-training industry dataset. The same reasoning can be applied which is described in the section of comparison of layers for sensor acceleration 4.1.2, that these columns are raw and can be easily predicted, unlike position columns. Overall accuracy is the same irrespective of number of layers in the network around 86%.

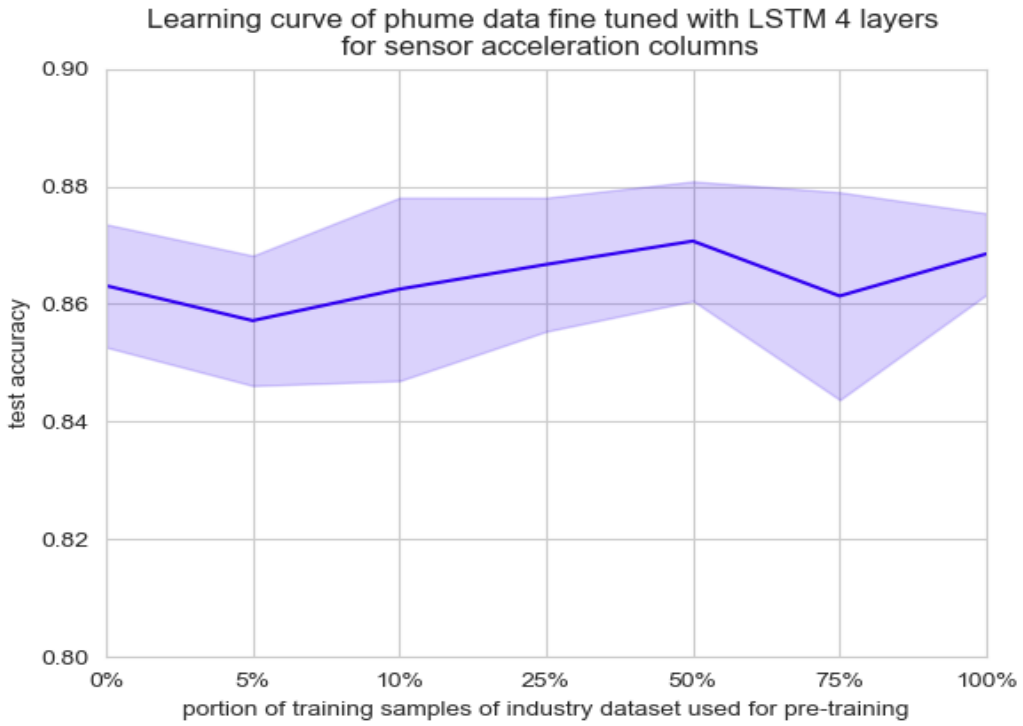

Figure 4.17: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 4 LSTM layers with sensor acceleration columns

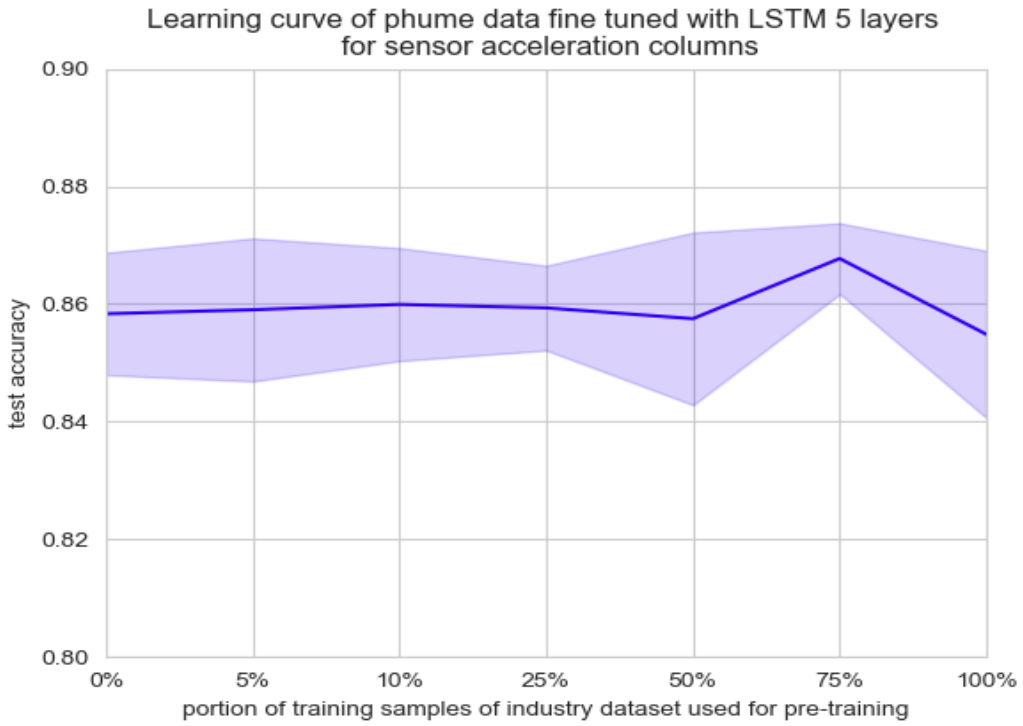

Figure 4.18: The learning process of the phume dataset is used completely for fine-tuning, but a portion of the industry dataset has been used for pre-training with 5 LSTM layers with sensor acceleration columns

4.3 Partial Fine-tuning

Fine-tuning the last layer or few is a very popular method that is applied in the Computer Vision field with many CNN models, as well as in transformers with BERT, GPT. The main idea is to replace the last layer and only fine-tune it when the data is not sufficient. The same approach is applied here with 5 layers of LSTM architectures, and the results are discussed further. The training process is performed on the position segments covering full body joints.

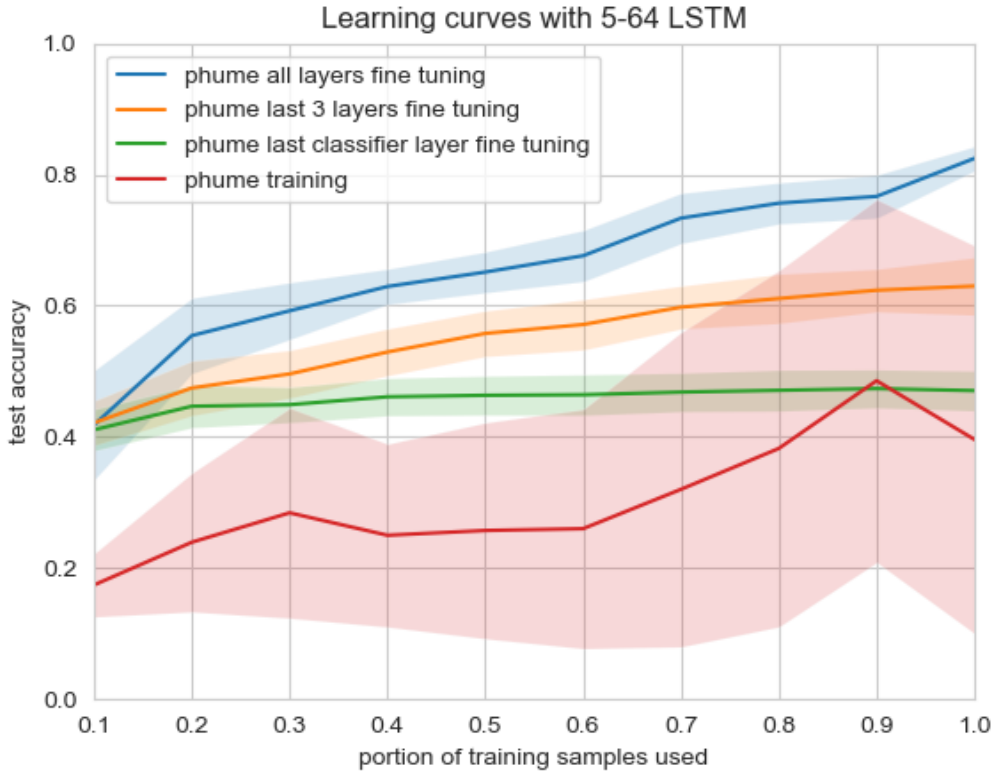

Figure 4.19: Learning processes with partially fine-tuned phume dataset for LSTM network of 5 layers

In Figure 4.19 by just replacing the last layer in the pre-trained network of the industry dataset and fine-tuning for the new set of classes, the results are not that significant, represented by the green curve. While fine-tuning, the network performed better than the training from scratch, shown by the red curve area, but does not contribute far better accuracy. The portion of the phume data is also varied with fine-tuning, but the progress is constant.

Similar to the process performed in the first part, in the next experiment, only the last 3 layers are fine-tuned, in which 2 layers are from LSTM and the last one is the classifier. The performance is better than fine-tuning only the last layer, represented by the yellow color curve. As the phume data used for the training process increases, the test accuracy advances to classify labels accurately.

The blue curve shows the results of the fine-tuning for the 5 layers architecture which is used previously in layers comparison. But this time the industry dataset is pre-trained with the same seed points because one of the seeds performed badly and test accuracy is low, which is also discussed previously in the first experiment setup of 5 layers of position data 4.1.1. Then fine-tuning is performed on the phume dataset randomly on new seeds and the results are better with less standard deviation, which clearly describes the goal of the thesis, that difference between mean curves are high with small standard deviation with no overlapping.

The concept of learning a few layers cannot be worked for this time-series sensors type of data. It probably requires large enough data to understand the pattern in a human motion or some feature engineering process needs to be performed to improve the accuracy.

4.4 Training with Bi-directional LSTM Network

The LSTM can also be applied in a bi-directional way. This concept is very helpful in the field of NLP, where understanding the context of long sentences is very helpful. Machine translation is the best example for these kinds of networks where the learning process starts from both sides, start and end. The learning parameters are double in the case of RNNs but in LSTMs, parameters grow exponentially. The reason for this is, LSTM is not a single neuron that just summaries the input using the function, but it has multiple gates and cell state, hidden memory which results in high parameters. Here parameters are near about 450K whereas a single-sided 5 layers LSTM network has 159K parameters. Performance of these networks and its results are discussed further.

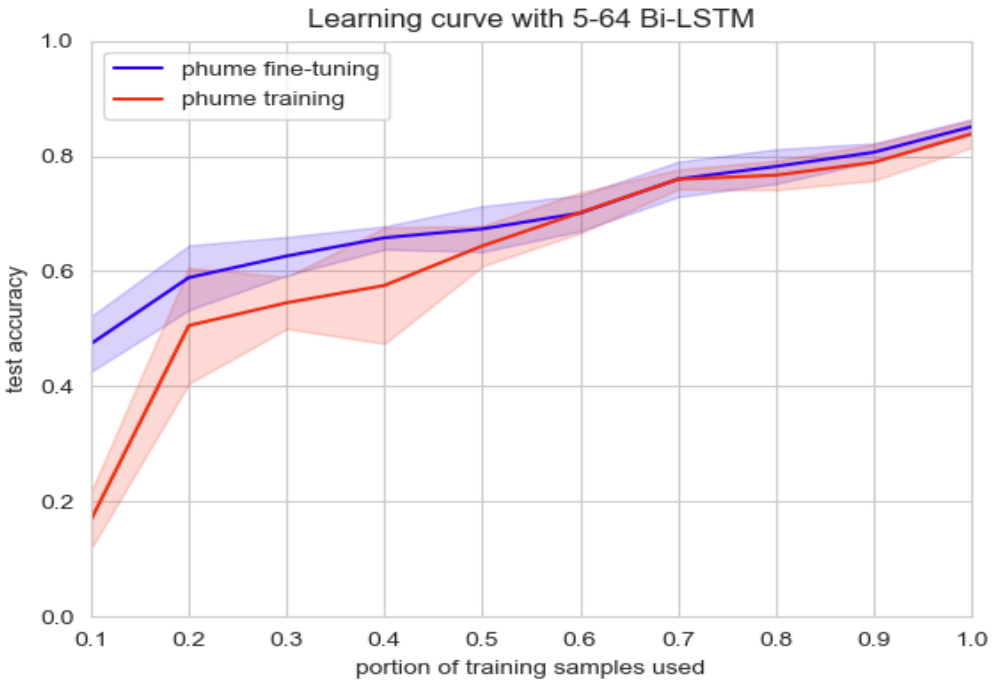

Figure 4.20: Learning process for Bi-LSTM network of 5 layers

The relative performance is the same for normal training and pre-training, as well as compared to a single-sided LSTM network in terms of test accuracy as shown in Figure 4.20. Since it is trained from start to end, there are many parameters, due to which the chances of retaining the appropriate pattern to classify labels also increase. Still, there is an initial improvement gained from the pre-trained network when the data is very less for training. As data increases there is no such noticeable difference.

4.5 Meta-parameter Tuning

In this section, unlike the experiments in other sections, the variable area is the meta-parameters such as batch size and learning rate. These two parameters are very crucial to tune the network to achieve better results. Since the direct effect has started to be seen from 5 layers network architecture, the same is used here in this experiment by keeping all other hyperparameters constant. The industry dataset pre-trained in the first experiment is used to fine-tune the phume data with varying batch sizes and learning rates in the second experiment. One of the reasons is the time required for training. The larger the network, the more time is required for the training process. The availability of resources is also a major factor when it comes to trying many combinations.

4.5.1 Batch Sizes

All the graphs are plotted in a single graph with a common x-axis label. The same process of training is applied here as for the initial experiments that portion of phume training samples used to see an effect on small and large datasets. The batch size is one of the meta parameters that need to try out many times to get the best performance for every layer and hidden neurons in the hidden layers of a network. According to batch size, the data is divided into batches of a size which helps not to process all data in a single run and make multiple iterations instead. The batch size and learning rate are directly related because, in the back propagation process, the loss is calculated using any loss function, and to minimize the loss, the gradients are calculated which is also explained in the training of neural network section 2.2. The results of these experiments are discussed further.

As the graph shows in Figure 4.21 that for lower batch sizes, the pre-training provides significant results. The reason behind this is, with a small batch size the data is not enough in a single iteration to understand the pattern and predict labels correctly. The loss is always high, and gradients cannot find the global minima in the optimization process. Generally, gradients decide in which direction the training process goes to find the minima, and learning rates are used to decide the step size after every epoch. The learning rate used here is 0.005, which is seems high for lower batch sizes.

As the batch sizes are increased, the simple training process is also showed the performance improvement increasing with a portion of training samples used for training. Pre-training also supported a little in case of a higher batch size. At

the initial stage, when less data is used for the training process, it can boost the performance and maybe also be used for online-based learning approaches. So, to learn more from data with smaller batch sizes, the learning rate should be small, due to which the iterations in a single epoch will be increased, and the test accuracy will be improved by training for high number of epochs.

The one more problem is observed here is the high variance with large batch sizes. To analyze this state more in detail, the last experiment with batch size 1024 is repeated four more times and results into learning curves of average of 50 repetitions. For the lower batch sizes with 32 and 64, the test accuracy maybe be more comparing to batch size 128 which is decided to be constant for all experiments when comparing layers. But the time required for the training process also increases as the batch size decreases and network layers increases. Even the training process for batch 128 is completed almost in 2 to 3 days, but with the batch size 1024, the training process gets completed within a day.

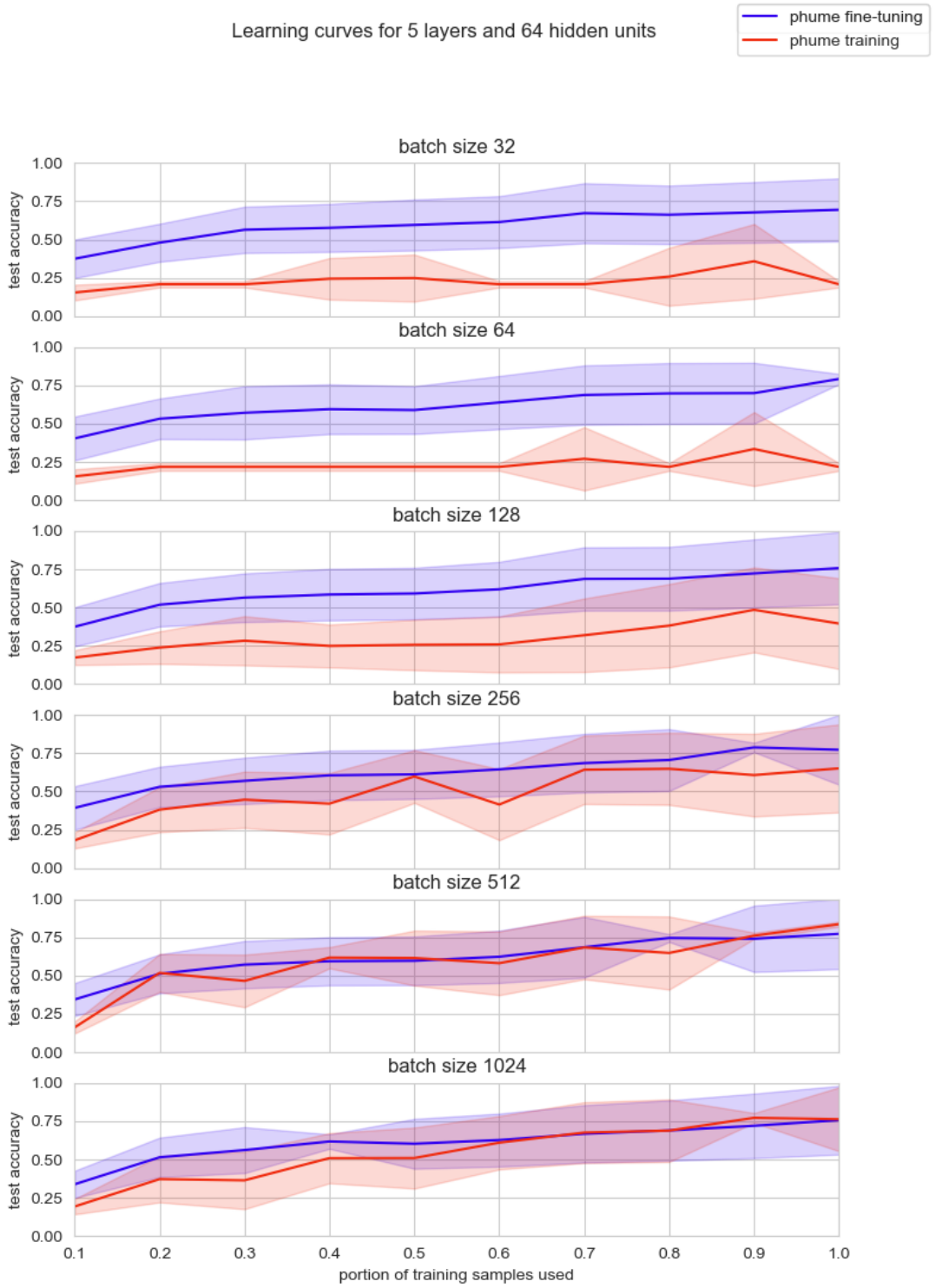

Figure 4.21: Learning processes compared for varying batch sizes for LSTM network of 5 layers

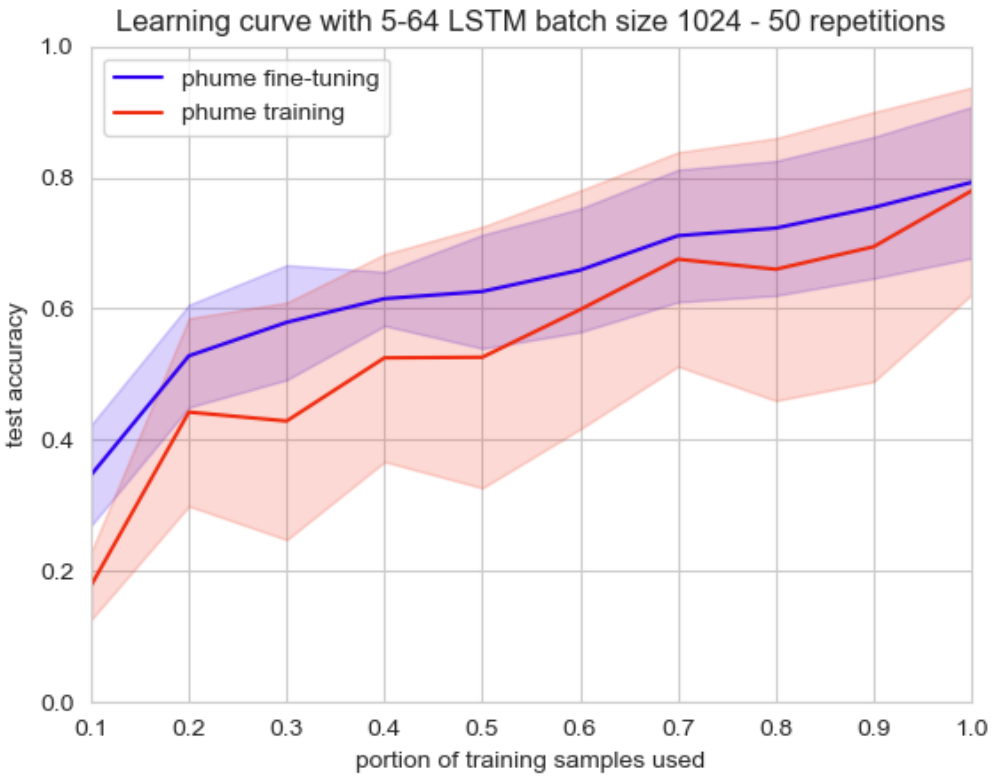

Figure 4.22: Learning process for batch size 1024 repeated for 50 times

The fine-tuning curve of phume data has reduced variance by repeating the training process shown in Figure 4.22. But there is no change in normal training of phume data. There exist a random effect and the plain training process fails to overcome this problem. On the other hand, transfer learning process is also useful to minimize the randomness and increase the overall performance.

4.5.2 Learning Rates

Similar to the last experiment, this experiment is conducted by varying the learning rates and keeping other parameters constant. The graphs are arranged in the increasing order of learning rates. The batch size used in this is 128. The condition discussed in the last section is if the learning rate would be lower for smaller batch sizes the normal training process would have been improved.

As results shown presented in the graphs of Figure 4.23, the advantage of pre-training is noticeable in higher learning rates. The same reasoning can be applied as for the batch size, that small learning rates should be used for smaller batch sizes. As the progress of curve seen in the first two graphs of learning rates 0.0001 and 0.001, pre-training is providing an initial advantage to achieve better accuracy when data available for training process is less i.e., 10% here. After that, both the curves are almost the same in the progress. Graphs with a learning rate of 0.005 and batch size 128 from the last section of batch sizes are with the same means generated from the same data. It is the same data mentioned in the first section when comparing the layers for position full-body columns. The standard deviation is high because one of the iterations when pre-training industry dataset, the results is poor and due to that, the effect is carry forwarded in this fine-tuning process. The reasoning and plausible solutions are discussed in the section 4.1.1.

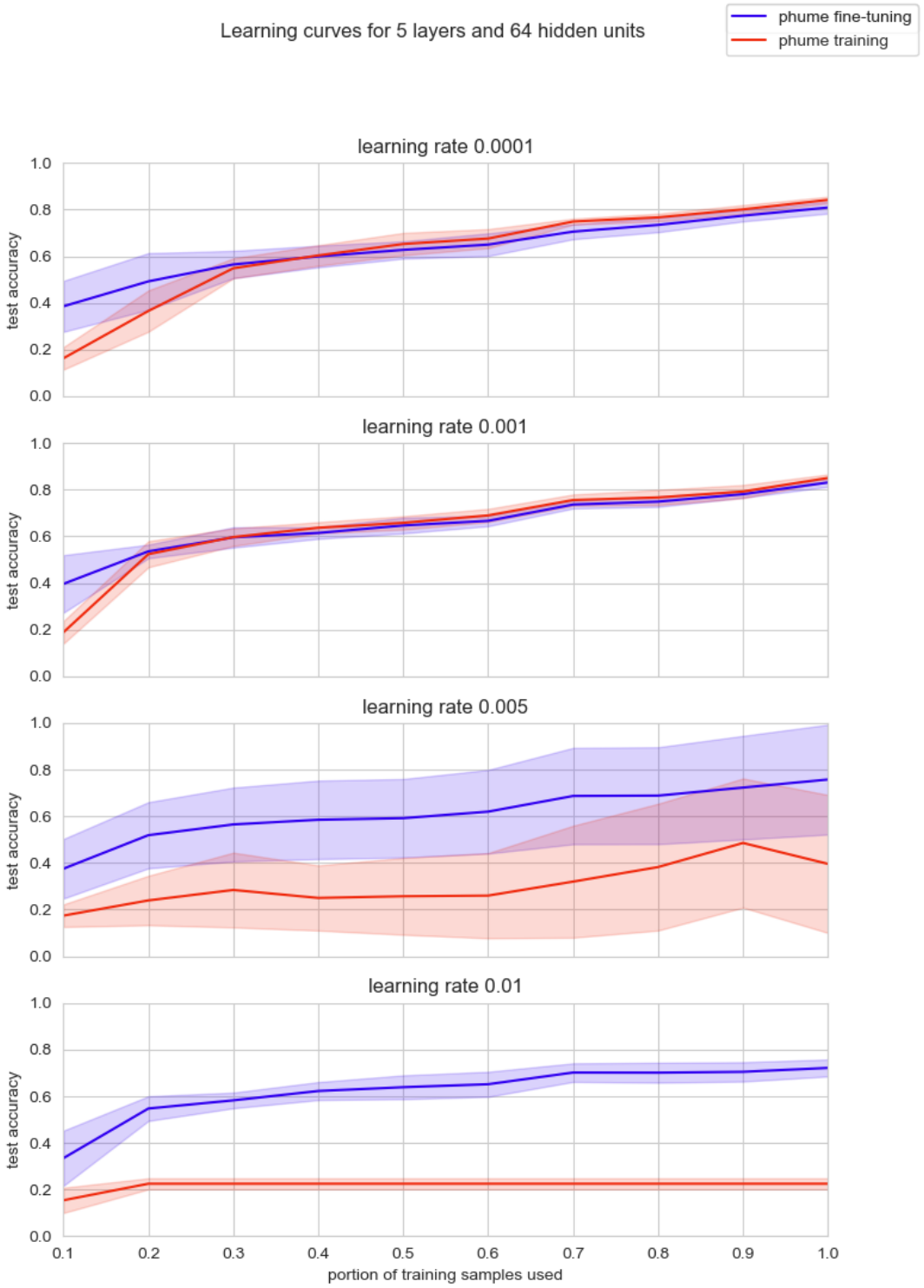

Figure 4.23: Learning processes compared for varying learning rates for LSTM network of 5 layers

4.6 Reverse Transfer Learning

In this section, the experiments are reversed, i.e., now the phume dataset is used for the pre-training and then the fine-tuning is done on the industry dataset. The hyperparameters used for this experiment are the same as used for initial experiments where the comparison between the layers has been done. In this experiment as well the different number of layers are compared. Generally, the industry dataset is large enough that it does not require any pre-training process to achieve better results. So, after 50% usage, the next step is with 75% and then 100%. Because after a certain point, it always has sufficient data to make a prediction with good accuracy. In the initial experiments, it is observed that for small networks like for 2 layers there is no significant improvement in the pre-training process, so here the experiments are started with 4 layers.

- **4 layers**

  There is not at all any significant improvement in the experiment with 4 layers in Figure 4.25. Both curves are predicting the labels very closely to each other. There is a slight improvement that can be seen at the start when 10% of data is used for the training process. The overall standard deviation is not large for all positions. It means other meta parameters are reasonable to make good progress in the classification.

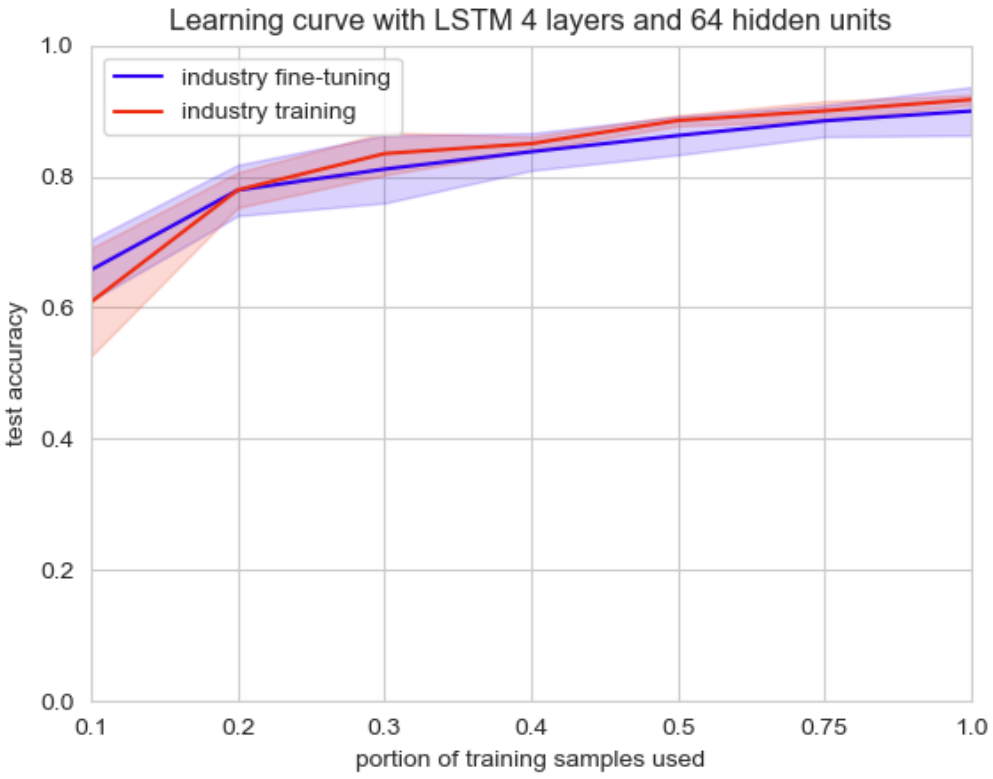

Figure 4.24: Learning process for reverse transfer learning for LSTM layer 4

- **5 to 8 layers**

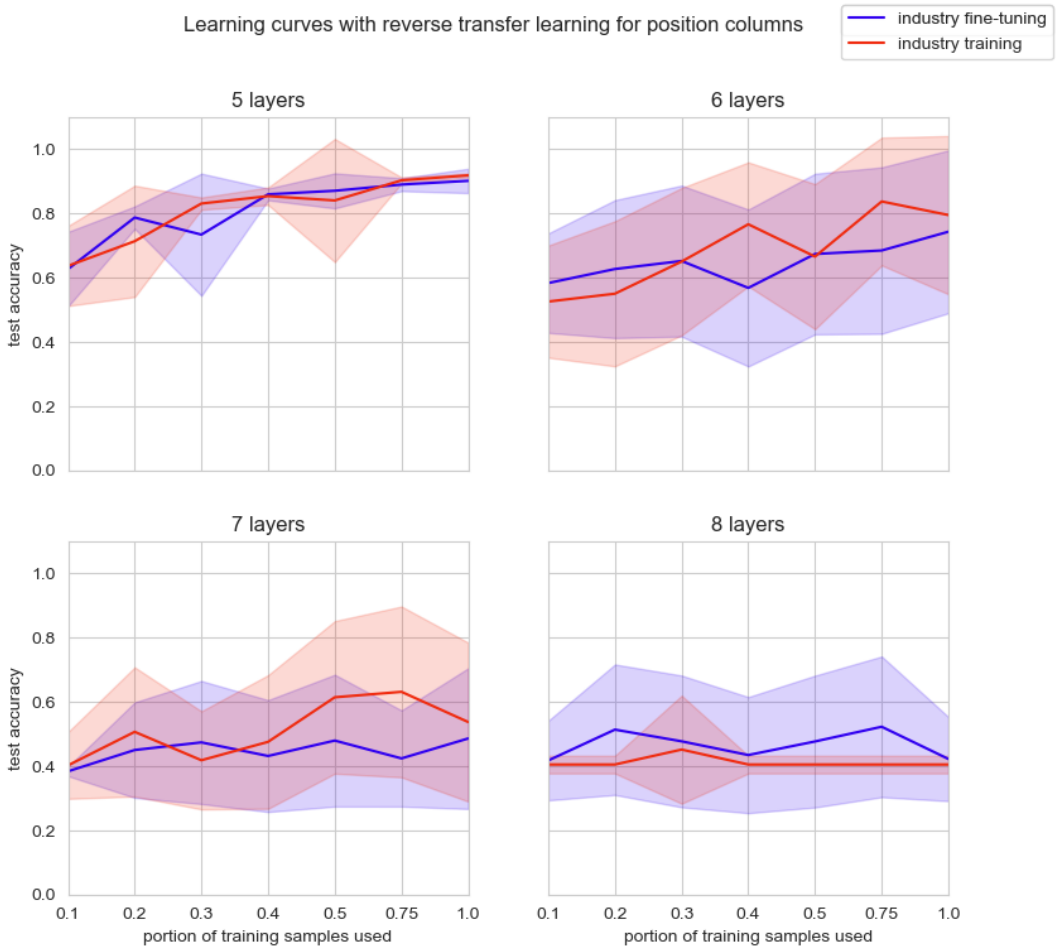

Figure 4.25: Learning process for reverse transfer learning for LSTM layers 5 to 8

In the experiments from 5 to 8 layers of Figure 4.25, the results are self-explanatory. There is no improvement with pre-training. There is no major difference at the start of the experiment and the main reason could be the 10% data of industry data is self-sufficient to predict the labels correctly. Pre-trained phume data does not contribute to performance improvement. At least for the layers with 5 and 6, there is some improvement with increasing data for training process, but the same is not observed for the layers with 7 and 8. The standard deviation for these high layers experiments is very high, such that many repetitions probably could fail to minimize it.

Chapter 5

Conclusion

Pre-training has been successfully applied to improve model generalization by analyzing values concerning IMU-based time-series data. In the next section, the detailed conclusion with performance comparison and metrics is described.

5.1 Conclusion

After analyzing many experiments and using two neural networks, LSTM and GRU, with many variations, it has been concluded that having a pre-trained network is the advantage with the multivariate time-series motion sensor dataset, especially for large networks like layers with 5 and 6 for LSTM neural networks. Results for these experiments are significant with the proper difference in the means with less standard deviation having no or less overlapping. The GRUs performed very well for very large networks, like for the cases with 7 and 8 layers. The optimization process succeeded compared to LSTMs but for 4, 5, and 6 layers the accuracy of LSTMs is better than GRUs. This comparison is for the full body joint Position columns. With Sensor Acceleration columns, the results are not significant maybe because they are raw sensor values, and it is easier to predict.

Moreover, the conclusion is for higher layered networks, pre-trained on the small dataset is very helpful and outperforms the training from scratch where a portion of industry datasets are used. The next point to consider is, with the multivariate time-series motion sensor data, the fine-tuning process should be done on all layers of the network because partial layers' fine-tuning does not give the required results. The bidirectional setup for the higher layer worked well on the initial stages when there is fewer data for fine-tuning the network. After a certain point, transfer learning does not contribute much too noticeable results.

In addition to this, the pre-trained networks are also mitigating the wrong selection of meta parameters. Sometimes, due to hardware and time constraints, need to select higher learning rates and/or batch sizes. In such cases, pre-training outperforms the training from the scratch. The results of multiple combinations of these parameters are very interesting to analyze, available in the sections above.

The last point to conclude is the general condition where the transfer learning is recommended that if the pre-training is performed on a small dataset compared to the fine-tuning dataset, the result is not useful. This is proved in the last experiment setup, where reverse transfer learning is executed, i.e. pre-training of phume dataset, and fine-tuned on industry dataset.

5.2 Future Scope

Firstly, the networks used here in experiments, are targeted to the family of RNNs in which LSTM and GRU are used. Just to make more variation, A version of LSTM, Bi-LSTM is also used to see a performance improvement. LSTM is the most popular and promising flavor of RNN, which works with a variety of data that has any kind of sequencing nature in any of the feature variables.

It is not always possible to have a large, labeled dataset, and it is a costly process to generate labels, therefore datasets can be used in an unsupervised manner. Instead of predicting a motion label for the classification task, the regression techniques can be used in a way that the next time step sequence is predicted. It currently means the window size is 60 which represents the motion sequence for a second, so the next 60-time window sequence is predicted and compared with the actual next sequence. Then the pre-training process can be used in this unsupervised way and then apply over the second task to fine-tuning.

The next variation in a neural network is the use of transformers. As the sequences are processed parallelly in the transformers, the speed of training will be increased. Applying multivariate time-series motion data in the form it accepts is a very interesting topic to research further.

The pre-training results are significant in some points which can be useful for online learning where the current recorded live data is used for predicting the data points. Personalized training is the best example described by the author in paper [13], where it outperformed the averaged static data, so using the pre-training process, it will be again interesting to analyze the results.

Bibliography

[1] E. Coupeté, F. Moutarde, and S. Manitsaris, "Multi-users online recognition of technical gestures for natural human–robot collaboration in manufacturing," *Autonomous Robots*, vol. 43, no. 6, pp. 1309–1325, feb 2018.

[2] V. V. Unhelkar, P. A. Lasota, Q. Tyroller, R.-D. Buhai, L. Marceau, B. Deml, and J. A. Shah, "Human-aware robotic assistant for collaborative assembly: Integrating human motion prediction with planning in time," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2394–2401, 2018.

[3] A. G. Marin, M. S. Shourijeh, P. E. Galibarov, M. Damsgaard, L. Fritzsch, and F. Stulp, "Optimizing contextual ergonomics models in human-robot interaction," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[4] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, may 2016.

[5] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418302136

[6] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[7] J. W. L. I. P.-A. M. Hassan Ismail Fawaz, Germain Forestier, "Transfer learning for time series classification," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, dec 2018.

[8] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015, www.cs.ucr.edu/~eamonn/time_series_data/.

[9] A. Sagheer and M. Kotb, "Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems," *Scientific reports*, vol. 9, no. 1, pp. 1–16, 2019.

[10] oliverguhr, "transformer-time-series-prediction." [Online]. Available: https://github.com/oliverguhr/transformer-time-series-prediction

[11] huseinzol05, "Stock-prediction-models." [Online]. Available: https://github.com/huseinzol05/Stock-Prediction-Models/blob/master/deep-learning/16.attention-is-all-you-need.ipynb

[12] J. Grigsby, "Multivariate time series forecasting with transformers," Oct. 2021. [Online]. Available: https://towardsdatascience.com/multivariate-time-series-forecasting-with-transformers-384dc6ce989b

[13] V. Losing, T. Yoshikawa, M. Hasenjaeger, B. Hammer, and H. Wersing, "Personalized online learning of whole-body motion classes using multiple inertial measurement units," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9530–9536.

[14] D. Faggella, "What is machine learning?" Feb. 2020. [Online]. Available: https://emerj.com/ai-glossary-terms/what-is-machine-learning/

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, dec 1943.

[16] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[17] S. SHARMA, "What the hell is perceptron?" Sep. 2017. [Online]. Available: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

[18] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: https://books.google.de/books?id=EoYBngEACAAJ

[19] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[20] S. Agarwal, "Comparison of machine learning methods for radaremitter identication," Sep. 2019, master's thesis from RWTH Achen University.

[21] J. Brownlee, "How to configure the learning rate when training deep learning neural networks," Jan. 2019. [Online]. Available: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[22] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[24] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *Journal of intelligent and robotic systems*, vol. 31, no. 1, pp. 91–103, 2001.

[25] H. Adil, "Human activity recognition using wifi channel stateinformation (csi)," 2021, master thesis at Frankfurt UIniversity of Applied Sciences.

[26] M. R. Mohammadi, S. A. Sadrossadat, M. G. Mortazavi, and B. Nouri, "A brief review over neural network modeling techniques," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*. IEEE, sep 2017.

[27] R. S. Milad Mohammadi, Rohit Mundra, "Stanford university cs224d: Deep learning for nlp." [Online]. Available: https://cs224d.stanford.edu/lecture_notes/notes4.pdf

[28] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1310–1318. [Online]. Available: https://proceedings.mlr.press/v28/pascanu13.html

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, nov 1997.

[30] Colah, "Understanding lstm networks," Aug. 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[31] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[33] R. Aggarwal, "Bi-lstm," Jul. 2019. [Online]. Available: https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[35] S. Kostadinov, "Understanding gru networks," Dec. 2017. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[36] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, oct 2010.

[37] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *International conference on machine learning*. PMLR, 2015, pp. 97–105.

[38] I. B. Arief-Ang, F. D. Salim, and M. Hamilton, "DA-HOC," in *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*. ACM, nov 2017.

[39] T. van Kasteren, G. Englebienne, B. J. Kröse *et al.*, "Recognizing activities in multiple contexts using transfer learning." in *AAAI fall symposium: AI in eldercare: new solutions to old problems*, 2008, pp. 142–149.

[40] J. Serrà, S. Pascual, and A. Karatzoglou, "Towards a universal neural network encoder for time series," 2018.

[41] A. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," 2016.

[42] J. Brownlee, "A gentle introduction to transfer learning for deep learning," Dec. 2017. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[43] P. Maurice, A. Malaisé, C. Amiot, N. Paris, G.-J. Richard, O. Rochel, and S. Ivaldi, "Human movement and ergonomics: An industry-oriented dataset for collaborative robotics," *The International Journal of Robotics Research*, vol. 38, no. 14, pp. 1529–1537, 2019.

[44] xsens, "Mvn biomechanical model," Apr. 2021. [Online]. Available: https://xsenstechnologies.force.com/knowledgebase/s/article/MVN-Biomechanical-Model-1605783181874?language=en_US

[45] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning and Data Mining*. Springer US, 2017, pp. 314–315.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[47] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," 2018.

Note: All the references mentioned above were checked on January 03, 2022.

Appendix - Public Access

Implementation work for experiments and results is available at the GitHub repository:

- **GitHub Link:**
  - `https://github.com/kshitijyelpale/master-thesis-pytorch-mvts`
- **Repository Structure:**
  - The `Documentation` folder contains the report, and LaTeX code of this report.
  - The `images` directory contains the images used in the report.
  - The `results-plots` directory contains the experiment results and graphs generated using the results.
  - The rest of the files are of deep learning pipeline to train the models.

```
master-thesis-pytorch-mvts
    |-- Documentation
    |-- images
    |-- results-plots
    |-- README.md
    |-- industry_fine_tuning.py
    |-- industry_pretrained.py
    |-- modules.py
    |-- phume_fine_tuning.py
    |-- phume_train_test.py
    |-- plots.py
    |-- utils.py
```