Embedded System Application for Motion Prediction in Fluorescence Imaging of Live Cell Processes

University of Turku
Faculty of Technology
Master of Science (Tech) Thesis
Robotics and Autonomous Systems
June 2025
Julián Camilo Páez Piñeros

Supervisors:
Adjunct Prof. Hashem Haghbayan
Prof. Juha Plosila

UNIVERSITY OF TURKU
Faculty of Technology

Julián Camilo Páez Piñeros: **Embedded System Application for Motion Prediction in Fluorescence Imaging of Live Cell Processes**

Master of Science (Tech) Thesis, 53 p.
Robotics and Autonomous Systems
June 2025

This thesis evaluates different algorithms for motion prediction in the context of live cell imaging using portable microscopy. The objective is to reduce both the time required for image acquisition and the resulting file size, by focusing only on areas containing relevant information. This approach improves image quality and data efficiency, making it suitable for low-power, embedded microscopy platforms where processing resources are limited. An often used solution is to integrate an X-Y motorised worktable that moves either the microscope or the sample in a preselected grid pattern. This allows for the system to capture a series of smaller images, which then can later be combined through image stitching to create a larger, high-resolution final image. However, in portable microscope systems, there are often limitations caused by the size, power consumption, and available processing capacity. These systems typically rely on embedded hardware with reduced computational performance. As a result, image acquisition becomes slower, which makes the observed motion of dynamic samples, such as living cells, appear faster. In these cases, it becomes important to predict and track the movement of the cells during scanning to ensure proper alignment and reduce data loss. When running on a desktop system, models like Decision Tree and Random Forest gave very low mean squared error (MSE) although Random Forest had a much longer training time. Simpler models like Ridge and Linear Regression were much faster to train and predict, but they had higher errors. Deep learning methods like LSTM and SVR had the worst performance in terms of time and resources. For example, SVR had an inference time of 134 seconds and the highest error values, showing that it's not suitable for low-resource systems. When these models were deployed on the Raspberry Pi 5 using TensorFlow Lite, the results also showed that lighter models were more efficient. For example, Linear Regression had a fast inference time and low CPU usage, while Random Forest and Gradient Boosting had better accuracy but required mor CPU usage. More complex models like CatBoost and SVR had higher MSE and CPU usage, making them less ideal for real-time use on embedded systems. Overall, Random Forest gave a good balance between accuracy and processing load, making it one of the best options for motion prediction in portable microscope setups.

Keywords: embedded systems, fluorescent imaging, live cell, machine learning, microscopy, motion prediction

Contents

List of Figures

List of Tables

Acronyms

R^2 Coefficient of Determination.

ANN Artificial Neural Network.

DL Deep Learning.

DT Decision Tree.

FoV Field-Of-View.

GUI Graphical User Interface.

HMM Hidden Markov Models.

KNN K-Nearest Neighbors.

LR Logarithmic Regression.

LSTM Long-Term, Short-Term Memory.

ML Machine Learning.

MLA Machine Learning Algorithms.

MLR Multiple Linear Regression.

RF Random Forest.

RFR Random Forest Regression.

RMSE Root Mean Square Error.

ROC Receiver Operating Characteristic.

RSE Residual Standard Error.

RT Regression Tree.

SVM Support Vector Machine.

SVR Support Vector Regression.

1 Introduction

Microscopes are an indispensable tool for scientific research. However, the process of collecting and analysing the acquired images requires effort, time, and domain expertise. Researchers have relied on both manual annotation and visual inspection to complete these tasks, which are not only repetitive but can also come with subjectivity and inefficiencies. Nevertheless, in recent years, the constant development of Machine Learning Algorithms (MLA) has improved nearly every aspect of technology, including microscopy. [1] The integration of this computational power into microscopy has enabled the automation of such tasks in a more accurate and efficient manner, allowing researchers to focus their valuable time on deeper research branches. Some of these tasks are presented here to serve as examples:

- **Object Detection:** Efficient identification and counting of individual specimens (such as, cells or bacteria) in an image. [2][3]
- **Classification:** Sorting different types of cells or bacteria groups (such as, human dermal fibroblasts, HeLa cells, Bronchial epithelial cells, muscle cells or stem cells). [4][5]
- **Tracking:** Following movement or growth of cells or bacteria over time (e.g., during drug treatment or infection studies). [2]
- **Feature extraction:** Measuring cellular features (such as, cell area, shape, or fluorescence intensity). [6]

Microscopy has usually not been limited by the requirement of having small equipment or using reduced space, but in some scenarios, such as space research, there is a need for more compact, accessible, and adaptable microscopes, sometimes even without human intervention. This situation has triggered the miniaturisation and automation of microscopical systems, in which these previously mentioned tasks must be completed in real time and with limited hardware resources. [7]

In this quest to miniaturise microscopes, the requirement of keeping characteristics comparable with bench top microscopes is also present. Some of these challenges include keeping the optical resolution as good as possible (how small of an object can the microscope detect) and Field-Of-View (FoV) large enough to capture the relevant information, e.g., cell movement. Otherwise, most of the gathered data might not be usable information.

An example of an environment where this is necessary is space applications. The costs of sending spacecraft and satellites to space increase dramatically according to their volume, weight, and electrical power consumption. So any equipment, including microscopes, has to be as small and light as possible. This creates a challenge because the hardware and the electronics inside such instruments must be as efficient as possible, even when having limited power capabilities and energy constrains.

On this same context, there is a limitation in the amount of information that can be downloaded from space, due to the distance involved and the low power that is available up there. And so, there is a need for the microscope to transmit only the most critical and interesting data, sending back as much information as possible but using as little as possible data as well. This can be achieved not only using compression algorithms, but also optimising the raw data generated in the first place.

The proposed solution for this is to count with a cell tracking MLA that can

focus on the cells, follow their movement closely, while predicting where the next position is going to be. This way, the microscope can capture as much as possible key raw data, to then be converted to essential information. This method seeks to be a stepping stone on the way to generating images that are just a fraction of the file size of a standard image, taken with the same microscope, using the maximum FoV and resolving the smallest optical resolution. This not only helps to reduce the data that needs to be sent back, but also makes it easier to turn that data into something useful.

1.1 Research Objectives

1.1.1 Primary Objective:

- To develop a predictive MLA for cellular motion using real-time data from in-situ imaging. This predictive MLA would be implemented on embedded systems to support the understanding of biological processes in non-conventional environments.

1.1.2 Specific Objectives:

- To develop computational models that predicts cellular motion using parameters extracted from live cell imaging.
- To evaluate and rank these developed computational models based on accuracy, efficiency, and fast while working with in-situ imaging data.
- To test the computational models performance and resource usage in different processing platforms.

2 Literature Review

2.1 Existing tools and platforms

The main aim of this review is to conduct a consultation on the current Machine Learning (ML) of cell identification, segmentation and tracking, to evaluate which MLs are available in portable systems containing embedded electronic systems, and to evaluate possible ways to collect the usable data to train the cell tracking machine learning algorithm. Only the most relevant work found is presented here.

2.1.1 EllipTrack

The first relevant work found related to this is the EllipTrack[2], an algorithm that not only can use ML to track cells by their nuclear markers, but also create *cell lineage trees*, following cellular division and differentiating parent cells from child cells. EllipTrack can be downloaded from. `https://github.com/tianchengzhe/EllipTrack`. It performs cell segmentation by first enhancing image contrast and applying blob detection to identify *nucleus-like* regions. It then uses active contour models to refine boundaries, followed by watershed to separate overlapping nuclei. Ellipses are fitted to represent nuclear shapes. For tracking, it uses a global algorithm to link cells across frames based on spatial and morphological features. A local correction module then adjusts tracks by evaluating cell behaviours like mitosis or disappearance, reducing linking errors. This two-step tracking approach ensures

accurate lineage reconstruction. Relevant to this work because it does segmentation and tracking of cells, based on the nuclei shape.

2.1.2 Portable Cytometer

Another relevant work presents a compact field-portable and cost-effective imaging flow cytometer. By using Deep Learning (DL) in near real-time, for detection of *Giardia lamblia* cysts in water samples, and it does not require the use of labels or fluorophores. The system is connected to a laptop, and is able to detect fewer than 10 cysts per 50 mL. This means that the device offers the potential for fast, automated monitoring of drinking water quality while running in with limited computing resources. [8] This device is relevant for this work since the authors present a portable solution that runs in-situ and on near real-time.

2.1.3 Aro Spot Finding Suite

A Graphical User Interface (GUI) was created in MATLAB, and as the authors say, it can be used to train a computational pipeline that can do segmentation by distinguish the labelled fluorescent molecules from background noise. The project is called Aro. [4] They use a supervised machine learning technique, random-forest implementation from MATLAB Statistics Toolbox, that utilizes a training set created by a first GUI, then trains the model with this and finally uses a second GUI for review and revision of the results. That logic can be useful for this work in case that the training data gathered is not labelled. Relevant for this work since, if a dataset not suitable enough for this application is found in the internet, then a GUI can be created to label a collected dataset to then train the MLAs.

2.2 Imaging Techniques and Architectures

2.2.1 Backlight LED patterns

A different usage of machine learning has also been studied, where the researchers use GUI algorithms to select the best pattern to use when flashing backlight LEDs on to a microscopic sample to form an image, seeking to *infer fluorescence image information from unstained microscope images.* And then, *show a consistent improvement in performance as compared to conventional microscope imaging methods.* [9] This is relevant to this work because it presents improvements in optical imaging resolution, which can be taken into account in this work or used for future work.

2.2.2 U net architecture

The paper introduces a MLA method that uses 2 GUI and works well with few training images. The authors propose a network with two parts: one that reduces image size to capture context, and another that increases size for accurate localization. They use strong data augmentation to make better use of annotated images. The network outperforms a previous sliding-window method in segmenting neuronal structures in electron microscope images. It also performs best in the ISBI 2015 cell tracking challenge for transmitted light microscopy. The ML model is fast, processing a 512x512 image in less than one second on a modern GPU. The researchers provide their full implementation and trained models online. [10] Relevant for this work to study the data augmentation methods and its possible implementation in this use case.

2.2.3 Deepsea

The paper presents a GUI based MLA model called DeepSea, trained to do segmentation and separately tracking of single cells in time-lapse microscopy images.

Time-lapse microscopy is pertinent because it captures how individual cells change over time. DeepSea is a trainable model that improves accuracy compared to previous methods. It works for both segmentation and tracking, helping researchers study cell behaviour over multiple time points. The authors demonstrate DeepSea's effectiveness by using it to analyse cell size regulation in embryonic stem cells. [11] Relevant to this work because it provides the full algorithm training online, alongside the training, test, and validation datasets of relevant greyscale and masked images of the cell nucleus.

2.3 Applications of ML in Image and Motion Analysis

2.3.1 User Motion Prediction

The paper discusses the safety challenges of Virtual Reality (VR) as its popularity grows, especially with the Metaverse concept. Although VR provides deep immersion, it can make users unaware of their physical surroundings, leading to collisions even within safety boundaries. The study aims to address this issue by analysing user motion and predicting unsafe movements. Researchers recruited 62 participants to play VR games and trigger specific events. They categorized user movements into three basic types and collected data through questionnaires, headsets, and controllers. After gathering three-dimensional motion data, they extracted movement range features and built a relational dataset. Statistical analysis was used to find links between user characteristics and motion categories. Finally, ML models were trained for multi-class prediction, and the safety boundary system was implemented in the VR environment to reduce the risk of physical collisions. [12] Although the paper focuses on predicting user movements in VR, its relevance for this work lies

in the shared methodology with cell movement prediction. Both involve spatial-temporal data, motion segmentation, and feature extraction to train ML models. The study's approach to building relational datasets and classifying motion patterns offers valuable insights for structuring similar models in microscopy. While the domains differ, the ML pipeline remains applicable across both human and cellular movement prediction tasks.

2.3.2 Machine Learning in Flood Prediction

The paper reviews the use of Machine Learning (ML) models in flood prediction, highlighting their potential to improve forecasting accuracy and reduce flood-related damage. It surveys over 6000 articles, analyzing 180 influential ones where different ML models were compared. This study is focused on the performance of various techniques, such as Artificial Neural Networks (ANNs), Support Vector Machine (SVM), Support Vector Regression (SVR), and Decision Trees (DTs), and evaluates such techniques based on metrics different metrics, e.g., Coefficient of Determination (R^2), Root Mean Square Error (RMSE), and computational efficiency. Four main strategies for improving prediction accuracy are identified: hybridization of ML models, data decomposition to enhance datasets, ensemble methods to increase generalization, and optimizer algorithms for model tuning. [13] The relevance of this articles is the way *Performance metrics* are used, for example, RMSE measures the average prediction error lower values mean the model is making more accurate predictions. And, *Computational efficiency* shows how fast and resource-heavy a model is, this is crucial in real-time prediction tasks executed in embedded systems.

2.3.3 Mineral prospectivity

The comparison done in this work studies the performance of different MLAs wich are for example, ANNs, RTs, RF, and SVMs. Those are applied to mineral prospec-

tive modelling, and the comparative is done according to different criteria, such as, accuracy in defining prospective areas, sensitivity to the estimation of hyper-parameters, sensitivity to the size of training data, and interpretability of model parameters, as stated by the authors. They also report results of applying these MLAs to *epithermal gold prospectivity mapping* in the Rodalquilar district of Spain. Where the RF algorithm performed better than the others.). The winner algorithm had more stability and robustness, even after modifying the training parameters, The ROC analysis showed better success rates. But otherwise, all the MLAs would be useful in different situations, for example if there is limited evidence of ore deposits. Additionally, the models RF and RT provide interpretable parameters useful to understand how geological factors might affect mineralization. [14]

2.4 Relevant Comparative Machine Learning Studies

2.4.1 LR and RF model comparison

The paper aims to design and compare predictive models using Machine Learning (ML) algorithms. It shows that the Logarithmic Regression (LR) model performs similarly to the Random Forest (RF) model in predicting Heating, Ventilation, and Air Conditioning (HVAC) system malfunctions. However, the study notes that the predictive model is still in its early stages and can be further developed with larger datasets and other ML techniques. The paper suggests that future research should explore additional methods for classification, training, and prediction to refine predictive models for industrial applications in Industry 4.0. [15]

2.4.2 A comparison of RFR and MLR for prediction

On this research, the authors compare two Machine Learning Algorithmss; Multiple Linear Regression (MLR) and Random Forest Regression (RFR) in predicting the concentrations of nine neurochemicals that are involved in different biological processes. The study found that MLR generally outperformed RFR, with higher Coefficient of Determination (R^2) values for MLR in six out of nine cases, indicating better prediction accuracy. For instance, variables such as ornithine and glutamate showed lower performance with RFR, while MLR provided more reliable results. The Residual Standard Error (RSE) for MLR was also lower in most cases, suggesting better model fitting. Despite the overall superiority of MLR, the paper concludes that RFR can still provide useful predictive value in certain situations. Therefore, MLR is recommended for predictions in neuroscience when dealing with similar datasets. [16]

2.4.3 An evaluation of ANNs, RTs, and SVM

Four MLAs are put on comparasion on this research, specifically: Artificial Neural Networks (ANNs), Regression Trees (RTs), Random Forest (RF), and Support Vector Machine (SVM), in mineral prospective mapping. The study focuses on their accuracy, sensitivity to parameters, training data size, and model interpretability. The results show that RF outperformed the other MLAs, offering higher stability, robustness, and success rates. SVM and ANN also performed well, but with more complex training processes. The RF model, in particular, was able to accurately map both prospective and non-prospective areas, while other methods showed biases. The paper concludes that RF is promising for mineral potential modelling, especially when data is limited, and that RT and RF can help interpret geological controls on mineralization. However, the performance of these models may vary for other datasets. [17]

2.5 Datasets

There were several datasets available to use publicly online, but the one to use was chosen based on the following criteria:

- **Temporal Structure:** Presents time-lapse images of cells moving in a medium in different contexts, suitable for forecasting.
- **Data Granularity:** There are more than just a couple of cells per image, which allows for fine-grained predictions.
- **Domain Relevance:** The images taken are presented in a greyscale format, this aligns perfectly with the user case of having a monochrome sensor, used in fluorescence microscopy.

A short analysis of the found available datasets cell data is presented next.

2.5.1 Cell Image Library

Offers a broad range of fluorescence microscopy images from different organisms and cell types. The site is simple to use, with basic search options. Is not very advanced when it comes to filtering by experiment type, and finding time-lapse sequences is a bit difficult unless they're labelled clearly in the description. Some image sets have multiple frames or z-stacks, but the platform does not specialize in temporal datasets, so time-lapse content is not completely available. [18]

2.5.2 Allen Cell Explorer

Focuses on high-resolution 3D fluorescence images of human stem cells, and it's polished and modern. Their datasets sometimes include live-cell imaging and time-lapse videos, especially for studying cell structures through the cell cycle. The search tools are advanced, allowing to filter by organelle, gene, or cell state. Also, images

often come with segmentations and downloadable videos. Is a good candidate, but images would need to be extracted from every separated video. [19]

2.5.3 Human protein Atlas

Focused on protein expression data in different cell lines using immunofluorescence. The main goal is to show where proteins are inside the cell, so the images are usually static snapshots, not time-lapses. The website is very useful for localization of specific proteins, with strong metadata and antibody info. But for studying dynamics or changes over time, it's not the right place since they don't really offer temporal datasets. [20]

2.5.4 Image Data Resource (IDR)

Rich resource full of datasets from published biological research. Many datasets include time-lapse fluorescence microscopy, and it's designed to host full experiments, not just final images. Multi-frame sequences can be found directly in the web client and download the original files. The search interface is advanced but somewhat complex. Still, for serious analysis or method development with time-lapse data, IDR is excellent as it includes detailed metadata in some cases. [21]

2.5.5 DeepSea

DeepSeas is made mainly for AI training and testing, with fluorescence microscopy datasets that often include time-lapse sequences and cell tracking. The images are usually paired with segmentations and ground truth labels. The goal here is to support deep learning tasks, so the format is very structured. As this research is trying to develop and test computer vision models on time-lapse data, DeepSeas is a good fit. The only downside is that the site is quite limited in the number of

datasets and doesn't offer much diversity in cell types or conditions, but can be used as a starting point for future work. [22]

2.5.6 Figshare

A general-purpose repository, so you can find all sorts of datasets, including fluorescence microscopy time-lapses—but it depends on the uploader. Some entries are detailed and well-documented, with zipped image sequences or videos, while others are just figures or PDFs. There's a basic search tool, and it can help to include "time-lapse" or "live-cell" in your keywords. Figshare is hit or miss: sometimes you find useful stuff, but often it's not organized for easy reuse of time-lapse data. [23]

2.5.7 Selected Dataset

The most suitable one found is "DeepSeas", that, as described by the creators [24], is "A deep learning model and a public large-scale dataset for cell region detection and tracking projects". Developed by the Department of Biomolecular Engineering, University of California, Santa Cruz (Shariati Lab). [25]

The authors shared on the website the original used dataset, alongside a Matlab program that is used to *"crop and label cell and subcellular bodies in cell microscopy images"* and a Windows software that *"is a user-friendly software designed to enable researchers to 1) load and explore their phase-contrast cell images in a high contrast display, 2) detect and localize cell and nucleus bodies, 3) extract some useful features from segmented cell regions, and 4) track and label cell lineages across the frame sequences. It employs trained DeepSeas models in the segmentation and tracking processes. Users can also manually edit the software productions using the edit options and then save them in their local systems."* This is an extraction from their own developers website. [22]

In this work, only the dataset is used and the segmentation MLA are used, but

since it is required for the system to run on an embedded system, then the provided end-user software is not of use for this application, hence the predictive MLA has to be trained from scratch, as this methodology section describes.

3 Methodology

This study is done using an **experimental research design (comparative machine learning approach)** methodology, this is to compare the efficiency and the power consumption of different machine learning algorithms. A quantitative approach is selected, and this means to get the results by collecting numerical data regarding the accuracy of every machine learning algorithm. Specifically, the interest of this research lies in the execution time, resource utilization and energy consumption, but on the selected embedded system.

3.1 Machine Learning Algorithms Studied

Since this task is mainly based on forecasting time-series-based data, the following relevant machine learning algorithms are evaluated, note that each algorithm is tested on the same datasets to ensure a fair comparison:

- **Logarithmic Regression (LR)**: This basic method is used for predicting continuous values, is classified under supervised learning in regression problems. It attempts to draw or find the closest straight line that is closest to the given data points using the least squares method to fit the data trend. Then, it uses weights for each input, this can be used to explain the relation between inputs and the value to predict. It is fast and easy to use. But, it only works well if the relation is close to linearity. If the data is more complex, this model

is not very good. It is used to compare with other models because it is basic and easy to understand. [26]

- **Ridge**: Another regression model, similar to Linear regression, but with some help to avoid overfitting, it also uses the total sum of differences in between the observed and the predicted values to the power of two, and uses using weights (or coefficients). But the addition is that it includes a penalty proportional to the square of the coefficients, which keeps the coefficients smaller but add a risk to overfitting. But if the data is not linear, it might present big errors. [27]

- **Lasso**: Similarly to Ridge but the penalty system is different. Instead of just shrinking coefficients, Lasso can actually make some of them exactly zero. This means it can automatically select features, which is very useful when dealing with many predictors. It helps simplify the model while also preventing overfitting. Hence, it is good for both regression and feature selection. This is good when having many variables, and uncertainty about which ones are useful. Lasso helps reduce the complexity of the model. But if some variables are very similar, Lasso sometimes could keep only one. [27]

- **Decision Tree (DT)**: This model makes decisions by creating a tree of rules. It is easy to understand and it works well with both numbers and categories. It does so by splitting the data set into regions, doing so by asking yes or no questions based on the input variables. Each internal node is a decision rule, and each leaf is a prediction. The tree keeps splitting the data until it fits well, but can result in overfitting if the tree is too big. It is sensitive with small changes in data. [28]

- **Random Forest (RF)**: Is called forest because it builds many decision trees. Random subsets of the given data set are used to trained each three. This

randomness helps reduce the overfitting of the previously discussed MLA. Then every three makes a prediction. The final answer is usually the majority vote (for classification) or average (for regression). This makes the model more stable and more accurate. It is strong with noisy data and it works well with many problems. The model can capture complex patterns and it is good with large datasets and noisy data, but it is harder to interpret than a single tree. [29]

- **Gradient Boosting**: Similarly to random forest, it also builds decision trees, but in a different way, gradually, just one at the time. Each three is meant to solve the errors of the last trained three. It is very good at learning difficult patterns. It focuses more on difficult examples, so is very powerful and flexible. However, it needs careful tuning, of parameters. For example, how many trees, how deep, and the learning rate. If not tuned well, it can lead to overfitting. [30]

- **Support Vector Regression (SVR)**: It tries to find a polynomial function as simple as possible that fits most data within a certain margin of tolerance from the true output. It can also use kernel functions to capture non-linear relationships. SVR is good when the dataset is small and clean, but it becomes slow and hard to tune when the data grows. Is not very interpretable. Additionally, it needs to be tested with many parameters to find the best model. It is good when high accuracy is needed but with few available data. [31]

- **K-Nearest Neighbors (KNN)**: This one is a very intuitive method, because it just looks at the K closest points in the training data, and averages their values (for this case of regression) and attempts to make a prediction to get a new point. It doesn't learn any internal model, so it is called a lazy learner. It is very simple and can work well if the data is small, clean and well-distributed

but it is slow with big data and can be bad if the data is high-dimensional. Still, performance depends a lot on the choice of K and the distance metric. [32]

- **XGBoost Regressor**: Fast, accurate, and handles missing values and categorical variables quite well. It does this by using clever techniques such as tree pruning, regularization, and parallelization to make gradient boosting more efficient. Still, it can be complex to tune, since it has many adjustable parameters. [33]

- **LightGBM Regressor**: Another framework based on XGBoost, but faster. It grows the trees in a different way, using a leaf-wise strategy and not a level-wise approach, making it suitable for large-scale datasets with many features. This allows it to reduce loss more quickly and use memory efficiently. It also supports categorical features natively and includes features such as histogram-based binning to speed up training. It is good when speed is important. But, like XGBoost, it needs tuning to work well. It is a strong choice when having a lot of data but looking for fast results. Its aggressive tree-growing strategy lead to overfitting a few times. [34]

- **CatBoost Regressor**: In the same was as XGBoost and LightGBM, CatBoost is a *boosting* model. But it is better with categorical data. It does not need to convert categories into numbers manually before training. It is designed to work well out-of-the-box, with good default parameters. It also avoids some of the problems with other boosting methods, like overfitting and sensitivity to order of data. Similarly to other boosting models, it takes time to train and tune the best parameters when needed. Training can be slow, especially in this case that the dataset is large. [35]

- **Long-Term, Short-Term Memory (LSTM)**: is another kind of neural

network meant to be used with data sequences. It can "remember" important things from earlier in the sequence and "forget" what is not relevant. This is something which regular neural networks cannot do. It has special units (called memory cells) that decide what to keep, what to forget, and what to output. This makes LSTM good for problems such as stock prediction, speech, language tasks or time series. Hence, it could prove useful for time-lapse images analysis as in this case. But, training a LSTM MLA can prove to be slow, and it needs a bigger set of data. It needs a lot of data and careful tuning. It is more complex than other models. [36]

- **Hidden Markov Models (HMM)**: A model that works statistically for sequence data. It assumes that the system has hidden states and the data seen is a result of those states. It is useful when understanding the process behind the sequential data is a must, as walking steps, weather changes, or for the case concerning this document, cell state transitions in time lapses. It works by estimating the most probable sequence of hidden states that produced the observed data. Based mostly in probabilities. HMMs are interpretable and can be trained even on small datasets. But they are limited when the data has complex patterns. HMM works well with small datasets and is easy to interpret. But it is not very flexible with complex or noisy data. [37]

3.2 Ethical Considerations

This study does not include human participants, so ethical risks are very low. Still, all experiments followed good AI practices. All models were tested in the same way, and results were reported clearly to allow others to reproduce the work.

3.3 Experimental Setup

3.3.1 About Hardware & Software

Hardware:

Training and preliminary testing was done on a laptop with good performance. The system had an **AMD Ryzen 7 PRO 8845HS** processor with 12 cores and 20 threads, and an **NVIDIA RTX 1000 Ada Gen** GPU with 6 GB of dedicated memory. It also had **32 GB of RAM** (SODIMM 5600 MT/s), which was enough for training all models without issues. To measure the energy use, a **USB ampere and voltage meter** was used to check the power consumption in real-time during training and inference (On the Embedded Systems).

Software:

All experiments were run using **Python 3**. The main relevant libraries used were:

- **Scikit-learn** to run classic MLA models, for example, LR, RF, and others.
- **TensorFlow** for training LSTM neural networks.
- **XGBoost, LightGBM, and CatBoost** for gradient boosting models.

3.4 Performance Metrics

3.4.1 Efficiency:

To compare all models fairly, different metrics were used:

- **Root Mean Square Error (RMSE):** to measure how accurate the predictions are.
- **Training Time:** measured in seconds.

- **Inference Time:** how long it takes to make a prediction (in seconds).
- **Inference Accuracy;** the average prediction error, measured in pixels.
- **Model File Size** how much space the trained MLA takes on disk storage.

3.4.2 Power Consumption:

CPU usage was measured during training and power usage was measured during inference:

- On a **Laptop** using the built-in hardware (CPU Usage).
- On a **Raspberry Pi 5** to test performance on low-power edge devices.

3.5 Experimental Procedure

1. **Dataset Preparation:**
    - Cell images segmented.
    - Paths extracted using a custom tracking algorithm.
    - Incomplete cell paths (with not enough steps) were removed.
    - The data was then augmented artificially by transformations to generate more training examples.
2. **Model Training:**
    - All machine learning algorithms mentioned in this work were trained using the same data and in the same hardware, to make a fair comparison.
    - For each model the final size of the trained MLA file were recorded.
3. **Model Evaluation:**

- Each model was tested on the original (non-augmented) dataset.
- Inference time, accuracy (in pixels), and power consumption were measured.
- To reduce random variation, each test was repeated **50 times**, and the average time was used.

3.6 Data pre-processing Steps

To ensure consistency and reliability in the experiments, the data set was pre-processed according to the following steps:

1. **Image selection:** The segmentation data set [22] includes 3169 monochrome images, with different cell types, where the nucleus is most prominent.
2. **Image masking:** These images are then segmented using "DeepSeaSegmentation" model, and, 3169 equivalent masked images are generated. But before attempting to segment the full data set, a smaller portion of this data set is used: it includes 245 already masked images, which are located in the *set 1 / cell masks* folder inside the *bronchial epithelial cell* data set.
3. **Finding contours:** These masked images are represented in a binary two-level system, that means, every part of the nuclei is *set* to value of 255 and everything else is set to zero (0). This is portrayed in Figure 3.2, the original image is shown in Figure 3.1.

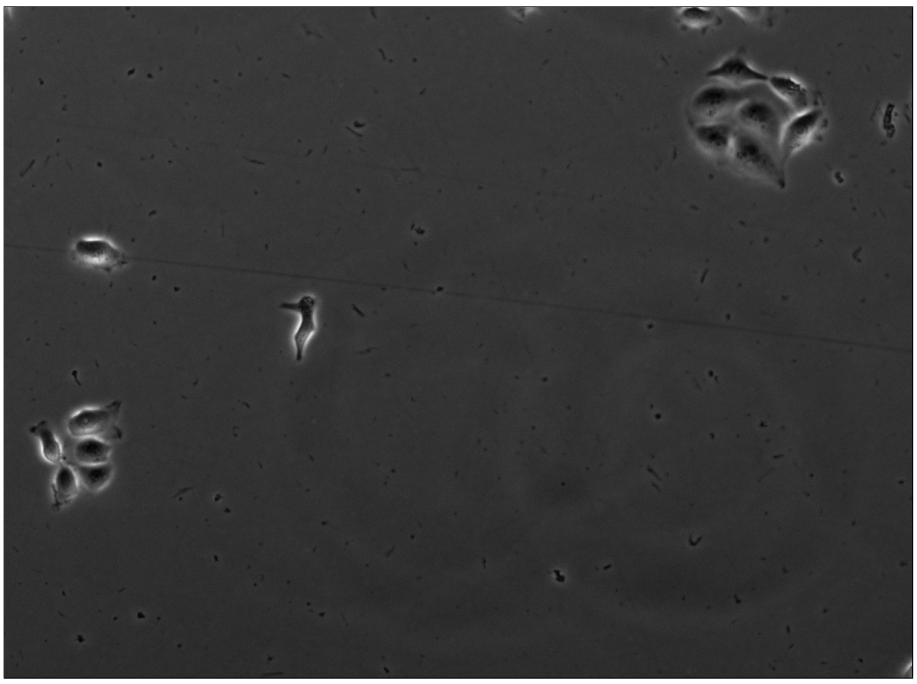

Figure 3.1: Original nuclei cell image sample. (DeepSeas data set[22])

Figure 3.2: Binarized masked nuclei cell image. (DeepSeas data set[22])

4. **Cell extraction:** Cell position is extracted from every image. 60 cells paths

are detected across all images in the selected data set portion.

5. **Cell cataloguing:** Cells are tracked according to how far they moved in the next image, the closest cell with the most similar area in the next image is selected to be the same cell in the next time step. Thus, a location list is created for every detected cell.

6. **Splitting paths:** Not every cell appears in all 245 images, so many cells do not have 245 recorded positions (Steps missing in the path). Some cells are only visible in 2 or 3 frames, for example, those that appear near the corners. To handle this, all cell paths were split into groups of 12 consecutive positions, and only the paths that were long enough were kept. After this process, 401 valid cell paths were obtained, a significant increase compared to the original non-normalized 60.

7. **Handling Missing Values:** Instances with missing or corrupt values are removed, and only 341 cell paths are classified as having enough and valid information.

8. **Verification:** These paths are shown in Figure 3.3, where the image borders are shown in a red rectangle. Note that each colour does not represent just a cell but rather a normalized cell path that can be from the same cell, this is why it looks more crowded when compared to Figure 3.2 and Figure 3.1.

9. **Data augmentation:** Additional time-based series are added by applying different operations to the original paths, such as, translation, vertical reflection, horizontal reflection, rotation, and scaling by different factors. This results in a data set 7 times bigger, with 2387 cell paths in total. Shown in Figure 3.4, as stated before, each colour represents locally a cell path. Note that the original paths are not present in Figure 3.4, but the same pattern can be observed with a scale factor of 2 (twice as big in coordinates).

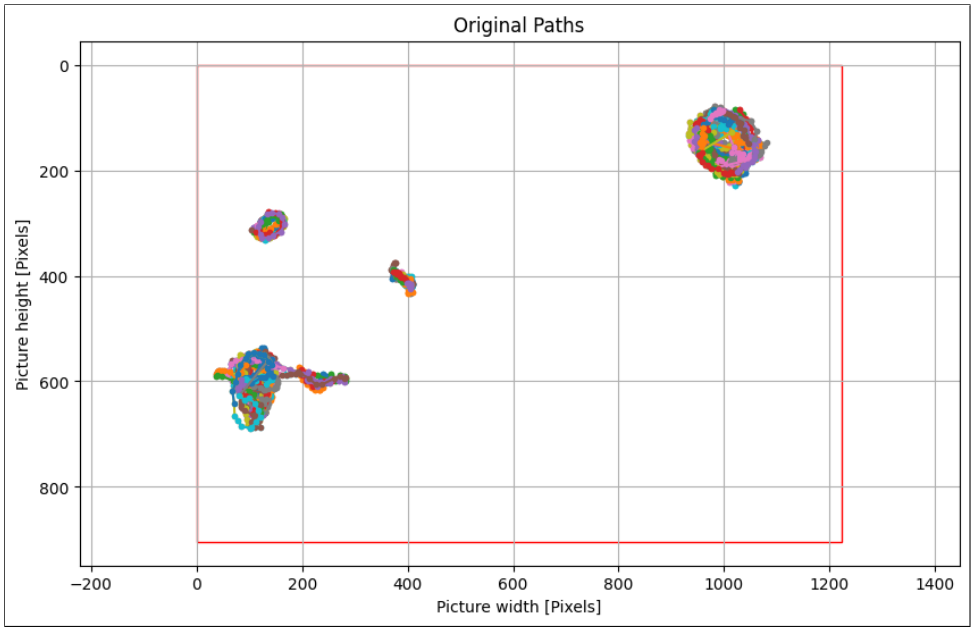

Figure 3.3: Original paths found.

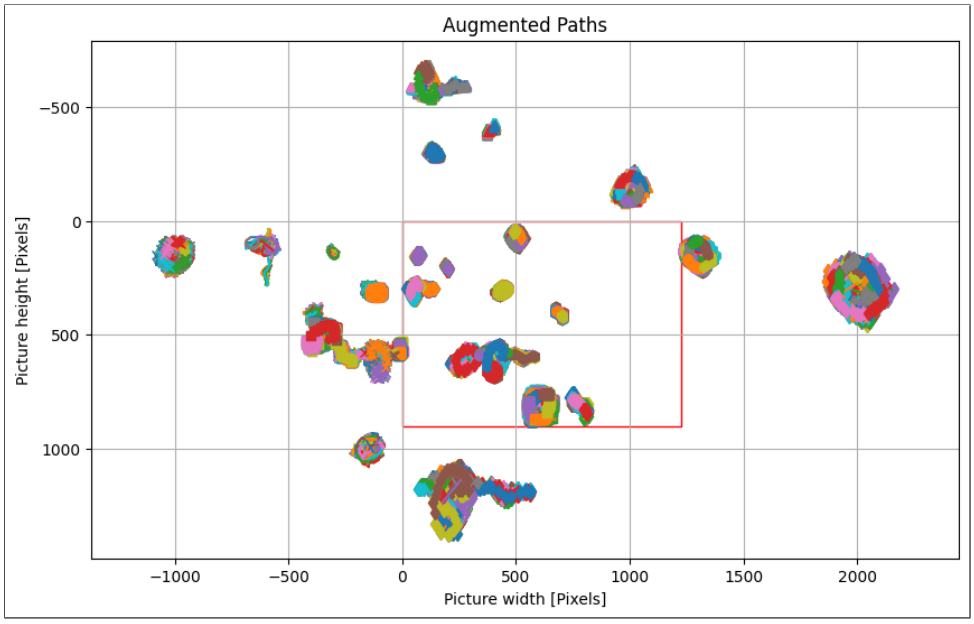

Figure 3.4: Augmented paths.

3.7 Proof-of-Concept: Random Forest Training

Once the data set was properly prepared, a RF MLA model was trained using 100 estimators. The training was done in several configurations:

1. **Split Method:** The original data set was separated in two, a training set and a testing set, in a 75:25 ratio respectively.

2. **Augmented Training Method:** In this set-up, the augmented data set (containing artificially increased samples through the methods mentioned in Section 3.6, item 9) was used to train the model. The original (non-augmented) data set was then used exclusively for testing.

The results of both approaches are presented as histograms as follows:

- **Figure 3.5** shows the error distribution error when the model was trained and evaluated using only the original data.

- **Figure 3.6** shows the error distribution when the model was trained using the augmented data and the prediction tested over the original data.

In both histograms, the X-axis represents the prediction error in pixels, calculated as the difference between the predicted inference value and the real value. The Y-axis shows how often each error value would in the inference done.

Based on the mentioned figures, it is clear that training with augmented data significantly improved the model's performance. The error distribution becomes much tighter, reducing from a wider range of approximately $(-15, 15)$ [pixels] and a RMSE of 25.248 to a smaller interval of around $(-2, 2)$ [pixels] and a RMSE of just 6.36. Most of the predictions fall within a five-pixel error when using the augmented data set, compared to a more spread-out error distribution when training only with the original data set.

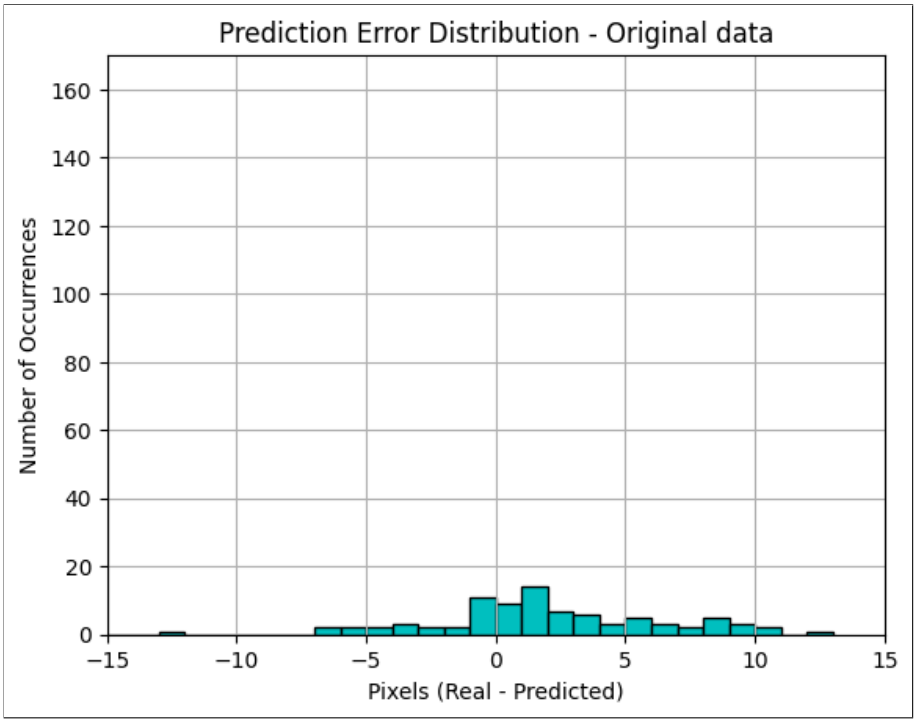

Figure 3.5: Original X Axis data RFMLA (histogram).

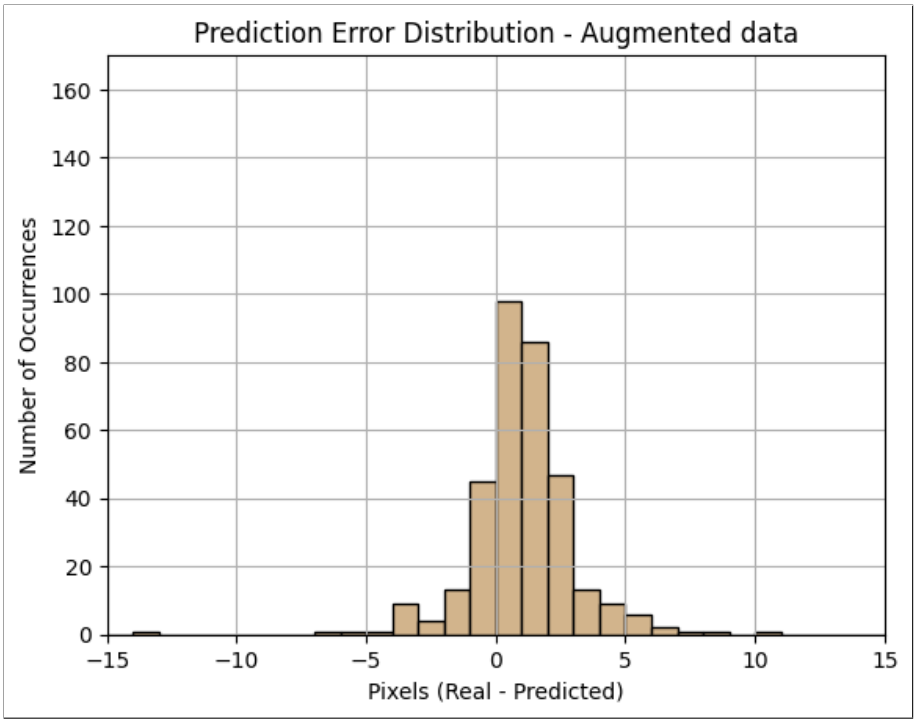

Figure 3.6: Augmented X Axis data RFR (histogram).

This reduction in RMSE, along with the tighter error distribution in the histogram, shows that training with the augmented data enables the model to learn more general patterns and improved its ability to predict values closer to that of the ground truth values (less error).

As of now, the model was trained only using the X coordinates of every cell path; hence, a prediction model for Y coordinates has to be created as well. Therefore, by using the augmented data only for training and the original data for inference testing, an RMSE of 0.305 is now obtained. Presented in Figure 3.7.

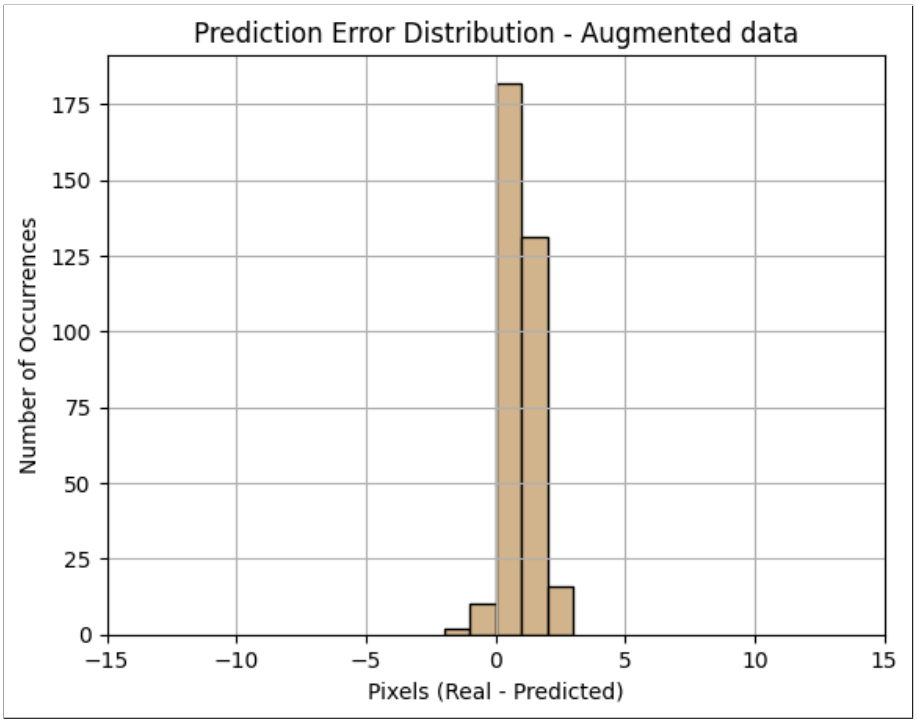

Figure 3.7: Augmented Y Axis data RFR (histogram).

Then, both X and Y coordinates of every path are mixed to study if it is possible to train a model that can predict both components (X and Y coordinates) of every cell path. The inference error for the mentioned scenarios is presented in an histogram as well, in Figure 3.8, with an RMSE of 0.616, Increasing for Y coordinates but decreasing significantly for X coordinates.

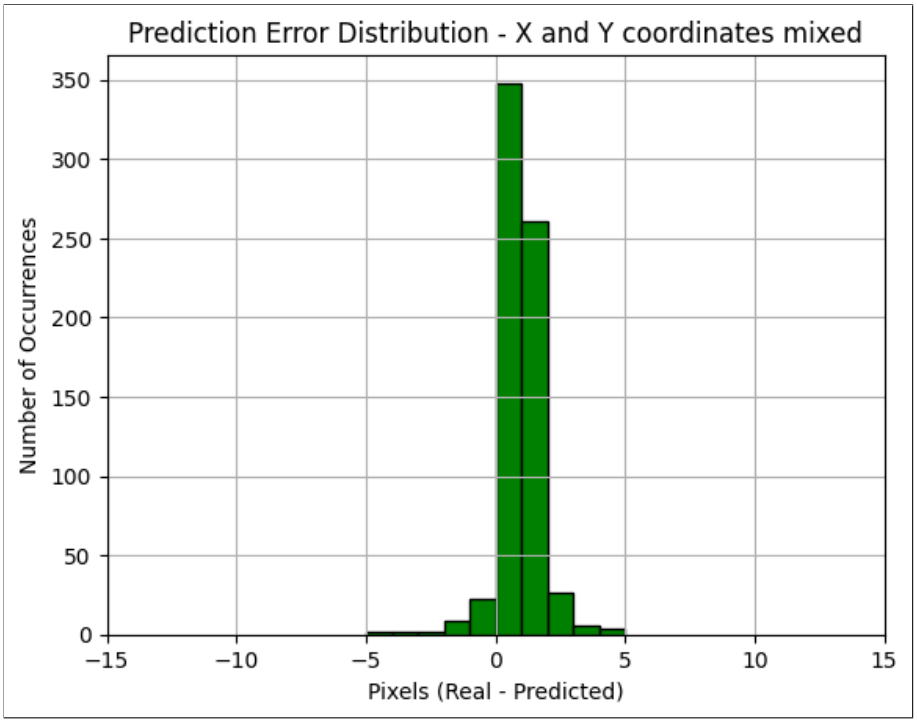

Figure 3.8: Augmented XY Axis data RFR (histogram).

Up until this point, it has been presented that the data set can be segmented, normalised, sorted, and used for prediction, and that it is possible to train a RF MLA using only a part of the data set, as shown in the previous Section 3.7.

4 Results and discussion

4.1 MLA Comparison Overview

Moving forward, several different MLAs introduced in Section 3.1 are trained using the full data set provided by DeepSeas[22], and following the same procedure used in Section 3.7. After pre-processing the data set a total of 8658 paths were found. These paths have more diversity in cell movements, cell types, movement directions, distances, and speeds. Also, the lighting conditions are different, and in some cases, the images are affected by substances such as dirt.

This results in the models trained to be more robust and able to handle a wider range of situations. A sample of these scenarios can be seen in Figure 4.1. Then, collected paths isolated from the complete data set are shown in a plot in Figure 4.2.

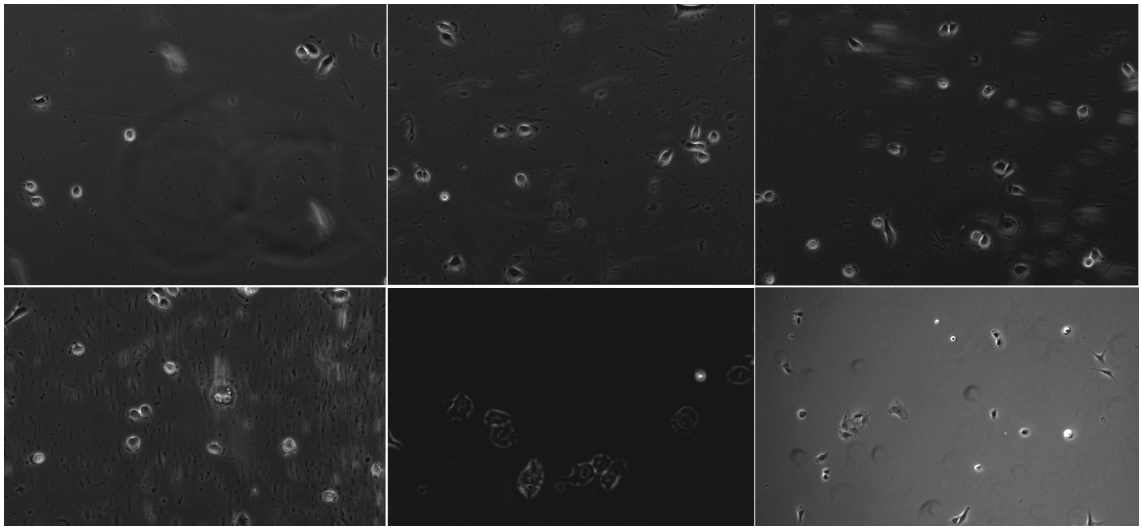

Figure 4.1: A few image samples from the full DeepSeas data set.

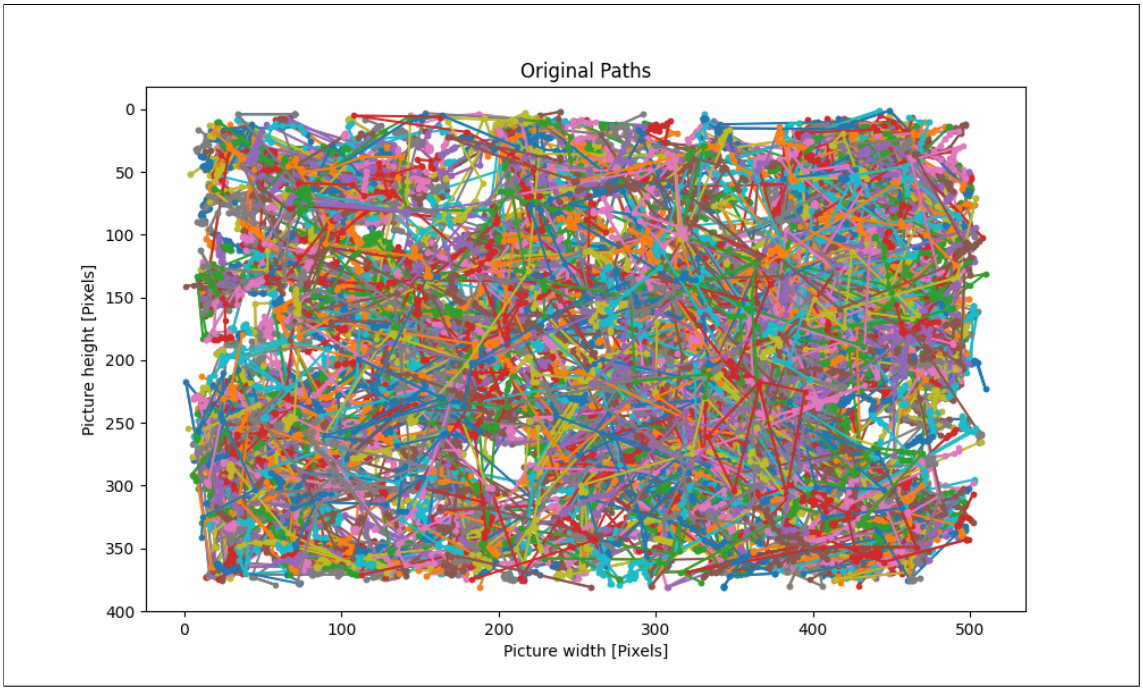

Figure 4.2: Paths found after pre-processing the complete data set.

4.2 Training and inference results (Laptop)

Then with the information ready, the MLAs mentioned in Section 3.1 can be now introduced and trained one by one. The mentioned results are also presented in a different way in Table 4.1 and Figure 4.16.

For all the evaluated models, the RMSE, training time, prediction time have been collected. After training the models, a performance index was calculated by normalizing the RMSE, training time and prediction time, giving them weights of 0.5 for each, and then adding them together. then sorting them according to the *Performance Index*, showing the smallest on top. This results were collected in Table 4.1. Please keep in mind that *Training (s)* and *Inference (s)* columns refers to time it took to complete the corresponding action.

4.2.1 Linear Regression

This is just more of a base line, but is also important to include it, and so, study if a simpler method is enough for prediction, or if using a more complex MLA actually makes sense. Figure 4.3 is the corresponding histogram result, it shows the prediction error after training a LR model with the collected paths from the full dataset.

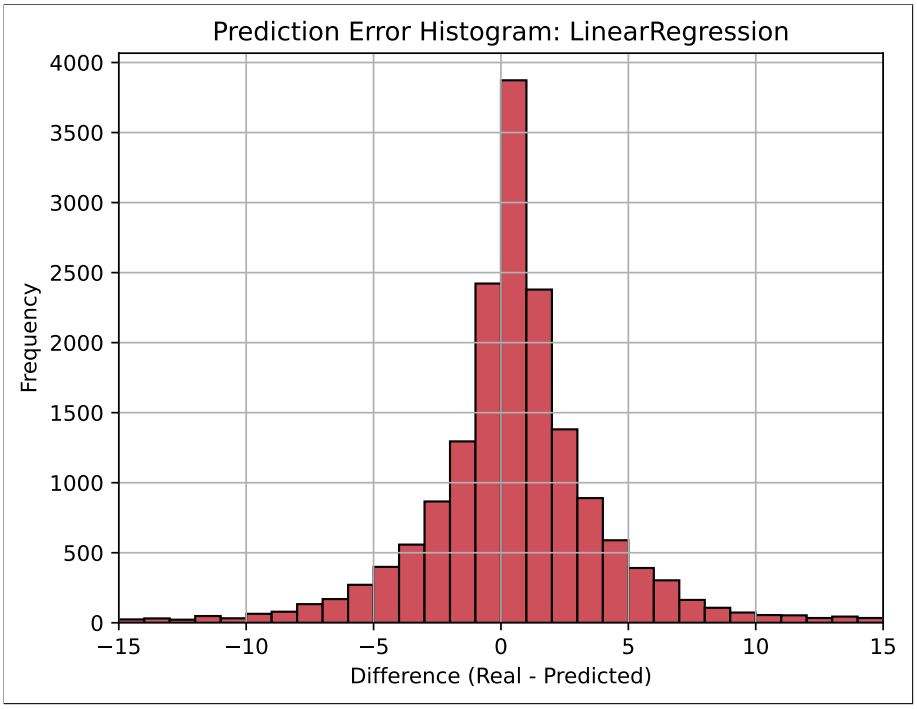

Figure 4.3: Full data set used on XY Axis coordinate data with LR (histogram).

4.2.2 Ridge

The corresponding result showing the histogram of collected data can be seen in Figure 4.4. Note that it looks similar to the histogram shown in Figure 4.3, at least regarding prediction results, both Ridge and Linear Regression achieved the same RMSE of around 56.91 pixels, but the important difference relies more in a shorter inference time. Ridge was slightly faster in both training and inference, with almost negligible times, making it the most lightweight model computationally. Its index value is also the lowest, meaning it's the most efficient overall, although not very accurate. These models are useful when fast, simple predictions are more important than precision.

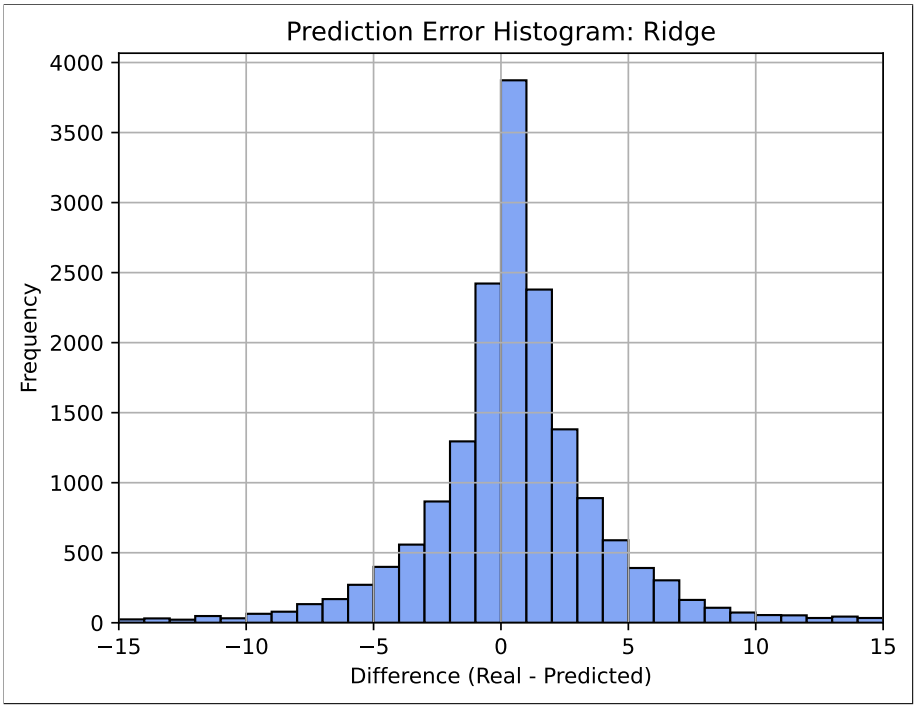

Figure 4.4: Full data set used on XY Axis coordinate data with Ridge (histogram).

4.2.3 Lasso

For this specific user case, Lasso found similar prediction values, with a small inference time as well, but a significantly longer training time (5.21 s) that the two previously presented regression alternatives. This makes it less attractive given the lack of accuracy improvement. Its high index reflects poor overall efficiency. Unless there is a strong reason to use feature selection via Lasso, simpler regressors would be better. The resulting histogram is presented in Figure 4.5.

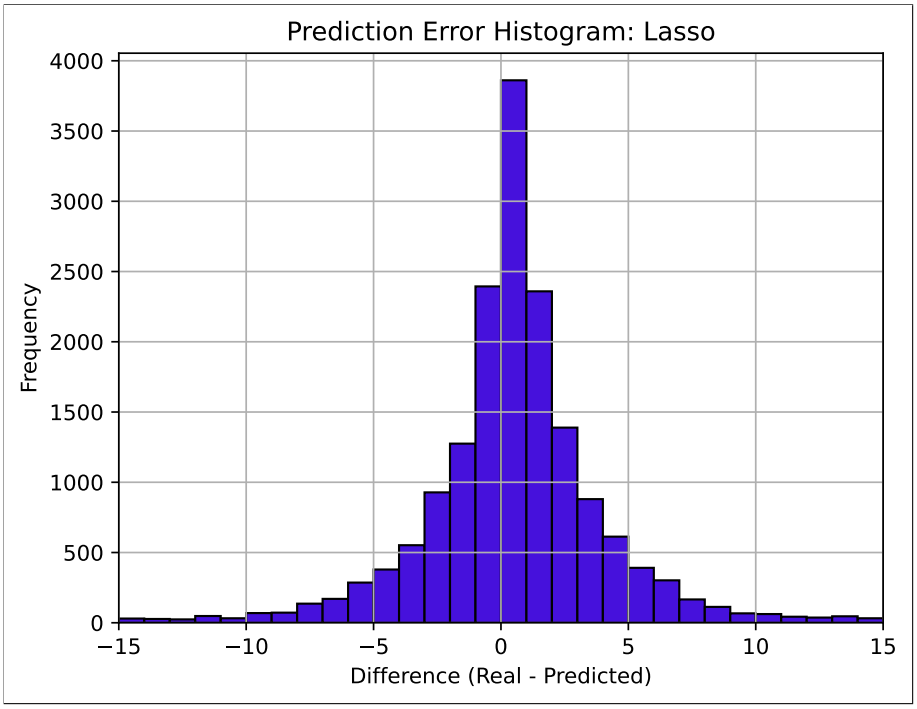

Figure 4.5: Full data set used on XY Axis coordinate data with Lasso (histogram).

4.2.4 Decision Tree Regressor

The present model had an exceptionally low MSE of 1.30, making it one of the most accurate models in this comparison. Its training time was just over 1 second, and inference was almost instantaneous. This high performance with relatively low computational cost results in a very low index. This model is highly recommended when accuracy is critical and training time is not a strict limitation. Resulting histogram is displayed in Figure 4.6.

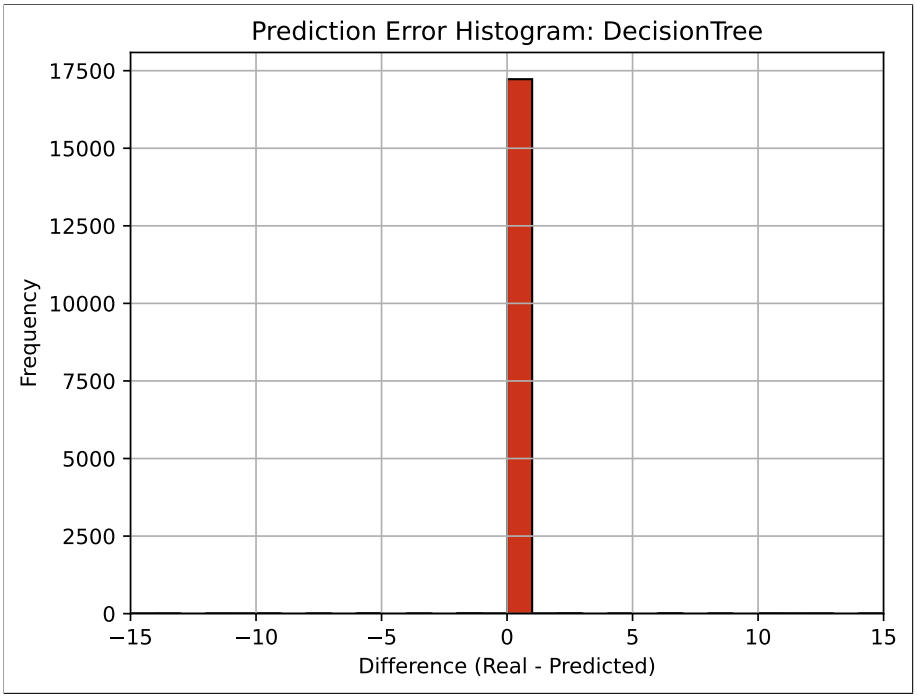

Figure 4.6: Full data set used on XY Axis coordinate data with Decision Tree (histogram).

4.2.5 Random Forest Regressor

Random Forest was the model used as a proof of concept, but now it was trained also using the complete data set. A small prediction error (2.31) is achieved with this MLA, showing excellent prediction accuracy. However, this came at the cost of very long training (almost 70 s) and a moderately high inference time (0.37 s). The index is high, indicating that although it's accurate, it's not as efficient as the previously analysed models, and the training time is significantly bigger. Resulting histogram can be exhibited in Figure 4.7.

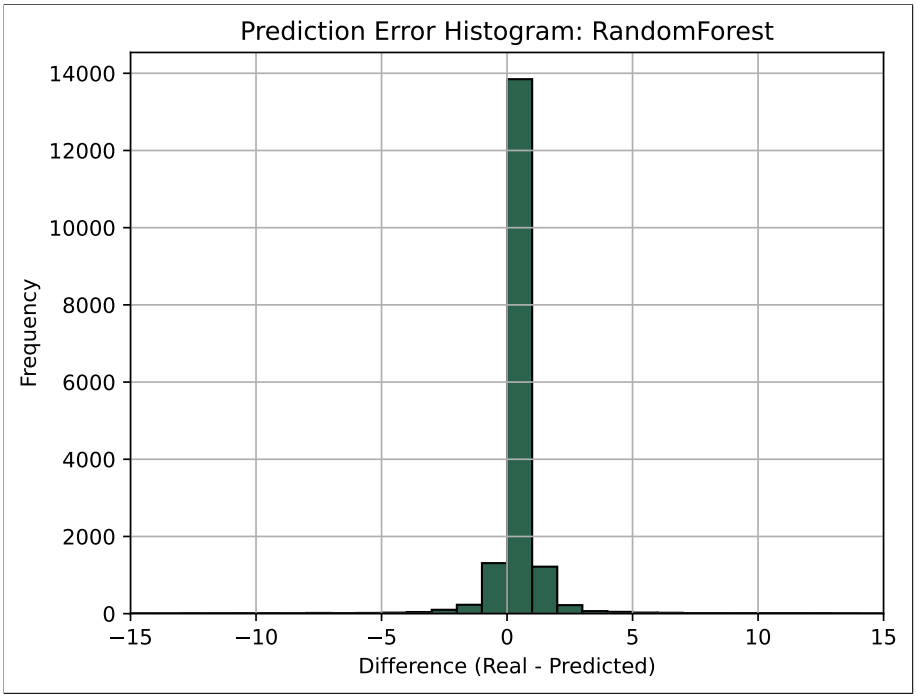

Figure 4.7: Full data set used on XY Axis coordinate data with Random Forest (histogram).

4.2.6 Gradient Boosting

This model had the worst MSE among all the tree-based models (60.76), along with a very long training time (17.84 s). Inference time was low (0.013 s), but overall, this method performed poorly in both accuracy and efficiency. It is not recommended in this case. The generated histogram after comparing real ground truth with the predicted values can be seen in Figure 4.8.

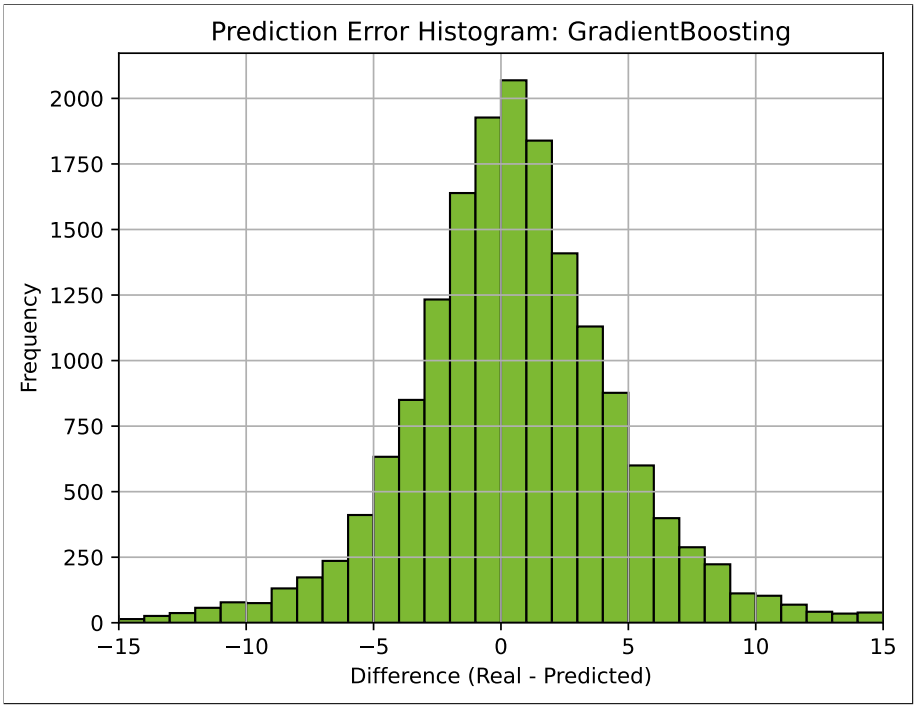

Figure 4.8: Full data set used on XY Axis coordinate data with Gradient Boosting (histogram).

4.2.7 Support Vector Regression

Since the training dataset is large, SVR had a high MSE of 68.52, and the highest inference time of all models (134.03 s). Training was also extremely slow (472.31 s). Both the training and inference times increased significantly. Although this suggests a potential increase in prediction error, the actual error was only slightly higher compared to the previously tested MLAs (68.52). Based on these results, SVR might not be a practical model for this case, as it is both slow and inaccurate. Corresponding resulting histogram depicted in Figure 4.9.

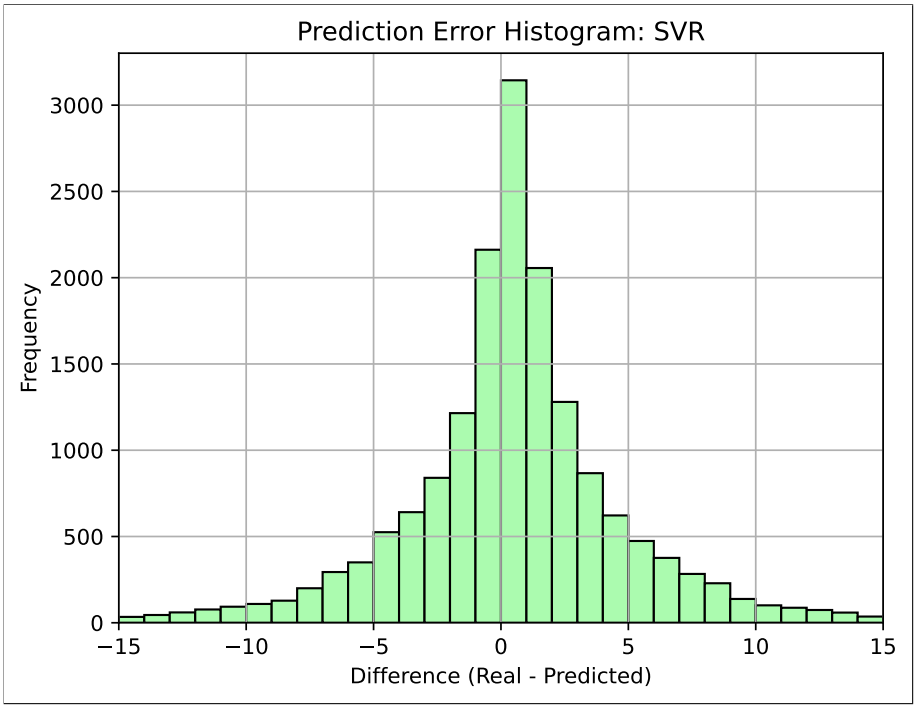

Figure 4.9: Full data set used on XY Axis coordinate data with SVR (histogram).

4.2.8 K-Nearest Neighbors

KNN had a decent RMSE of 30.84, but inference time was very high (0.65 s), which is typical for this algorithm since it needs to search through all training samples during prediction. The training phase was fast, but due to the long inference time, its index was higher, reducing its practicality for real-time applications. Corresponging histogram generated after comparing ground truth and inference is depicted in Figure 4.10.

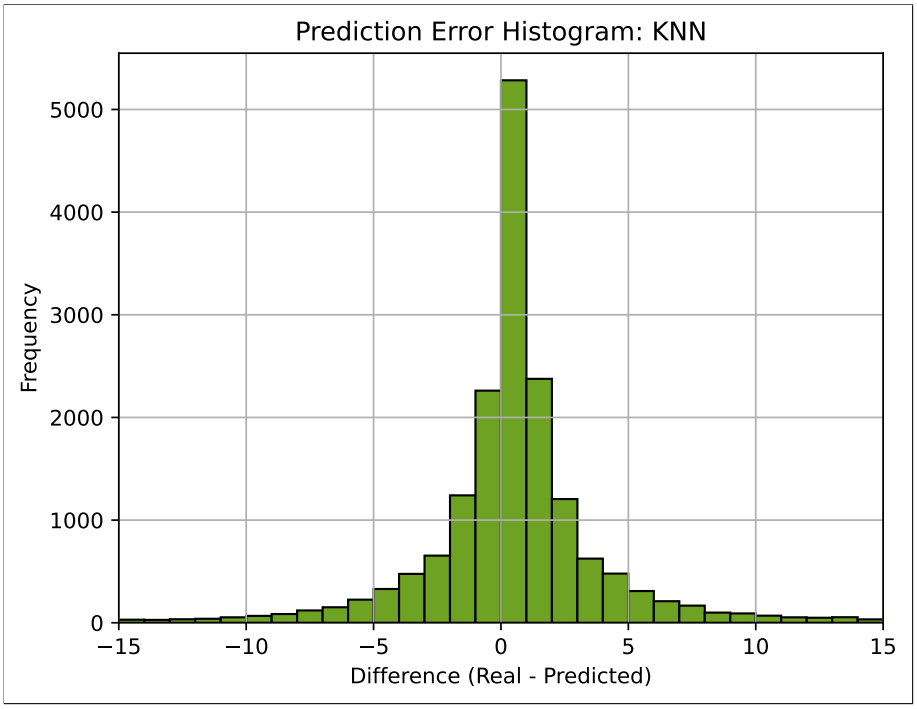

Figure 4.10: Full data set used on XY Axis coordinate data with KNN (histogram).

4.2.9 XGBoost

Performed significantly better in terms of accuracy with an RMSE of 43.73. Its training time was higher (0.47 s), but inference was still very fast. Its overall index is still low, which shows it is a good trade-off between performance and computation. This model might be a good default choice when moderate training time is acceptable, and accuracy is important. Histogram can be seen in Figure 4.11.

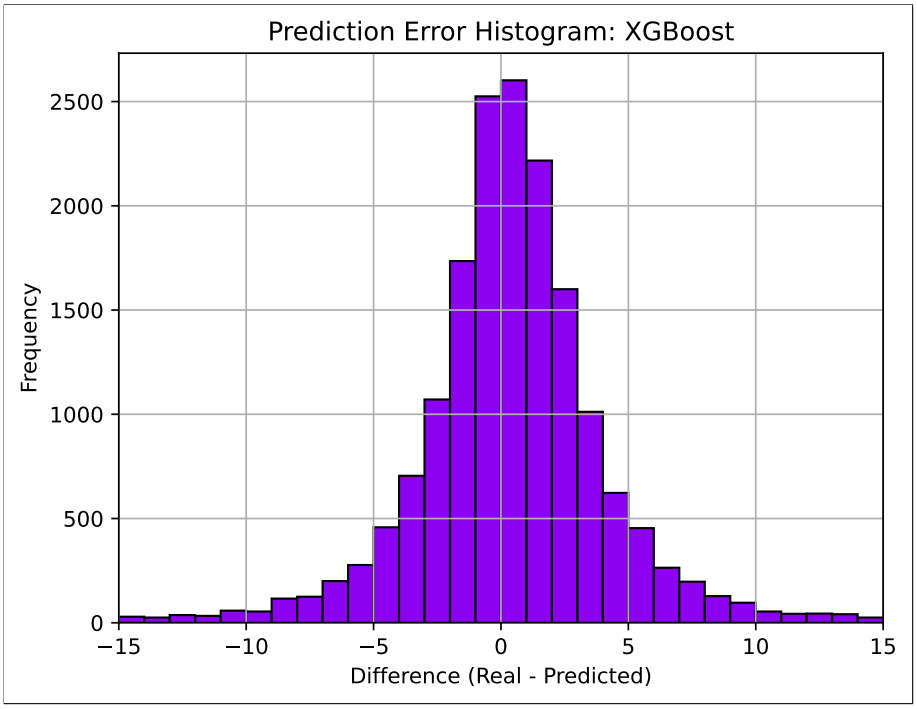

Figure 4.11: Full data set used on XY Axis coordinate data with XGBoost (histogram).

4.2.10 LightGBM

This MLA had slightly worse performance than XGBoost (RMSE of 54.23) and required more time for inference (0.03 s). Even though it trained slightly faster, the lower accuracy and higher inference time make it a bit less efficient. Histogram found in Figure 4.12.

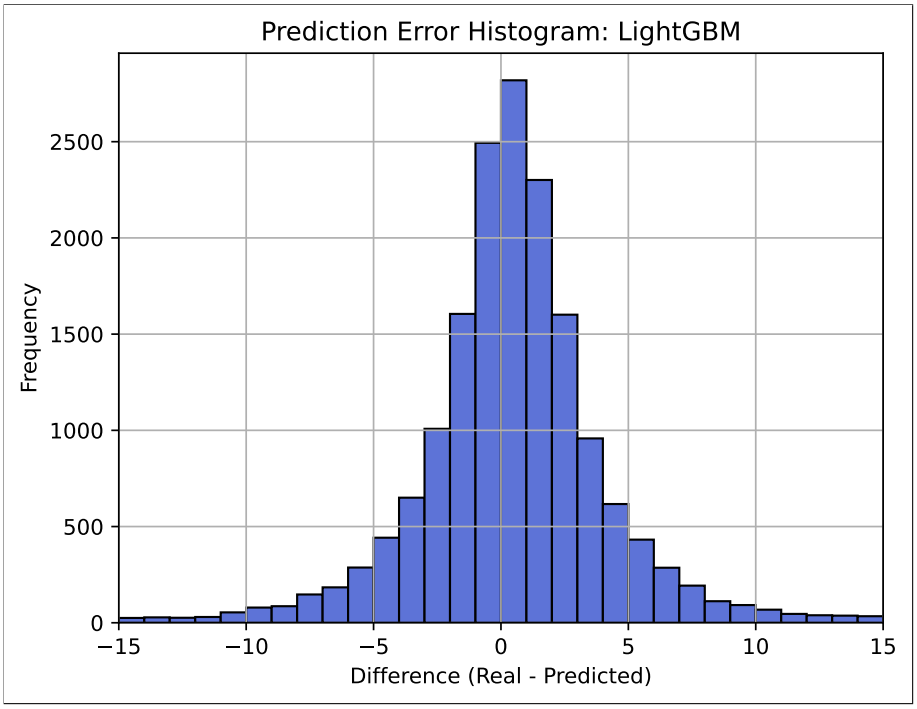

Figure 4.12: Full data set used on XY Axis coordinate data with LightGBM (histogram).

4.2.11 CatBoost

CatBoost performed moderately well, with an RMSE of 52.61 and slightly higher inference time (0.06 s). However, its training time was quite long (8.73 s). This makes it less efficient in this context, although its automatic handling of categorical data might be useful in other situations. Its index shows it's becoming less practical compared to XGBoost or DecisionTree. Resulting histogram presented in Figure 4.13.

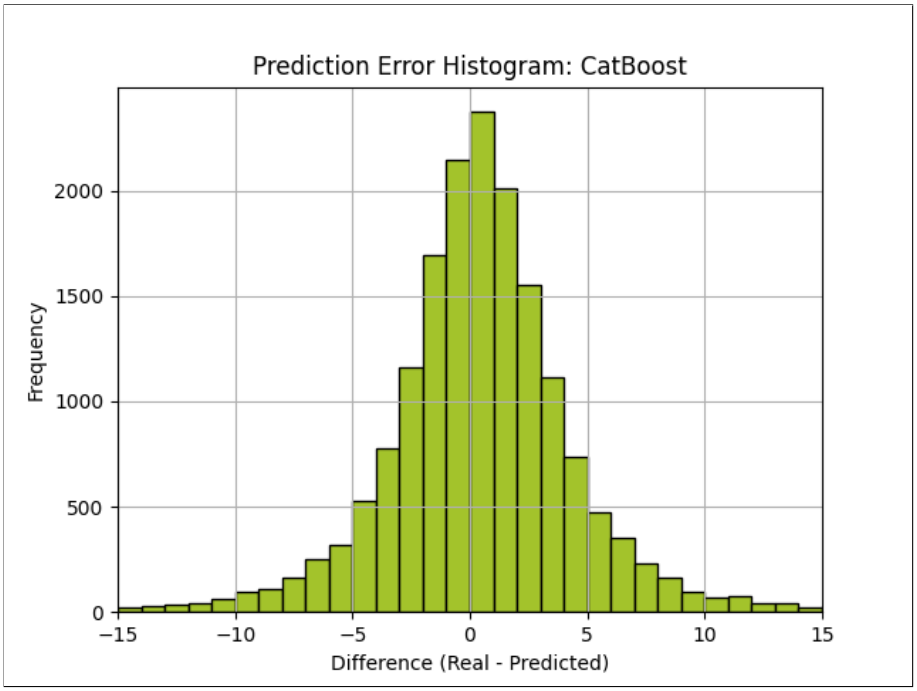

Figure 4.13: Full data set used on XY Axis coordinate data with CatBoost (histogram).

4.2.12 Long short-term memory (LSTM)

The LSTM model had the second highest RMSE (70.02), which is surprisingly poor given the extremely long training (112.81 s) and inference time (1.23 s). Its index is also high, meaning the model is neither efficient nor accurate in this setup. Histogram portraying the difference between the real (ground truth) value and the inference value is presented in Figure 4.14.

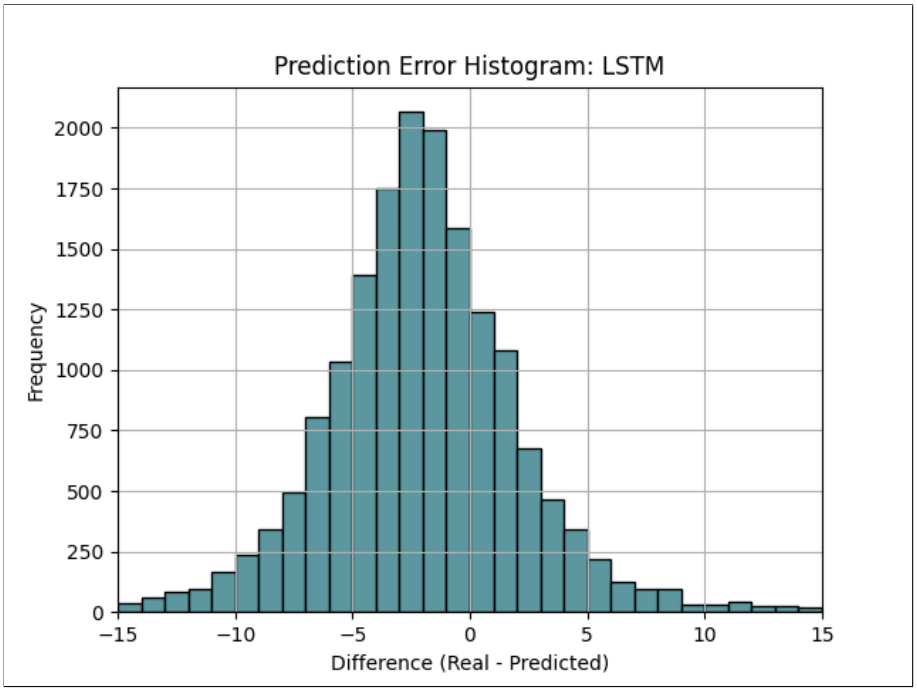

Figure 4.14: Full data set used on XY Axis coordinate data with LSTM (histogram).

4.2.13 Hidden Markov Models (HMM)

HMM had an extremely large RMSE of over 66000, making it completely unusable in terms of accuracy. Training and inference times were not excessive, but due to the bad results, this model is not a valid choice. Its index is also very high, confirming poor efficiency. The best obtained histogram is shown in Figure 4.15.

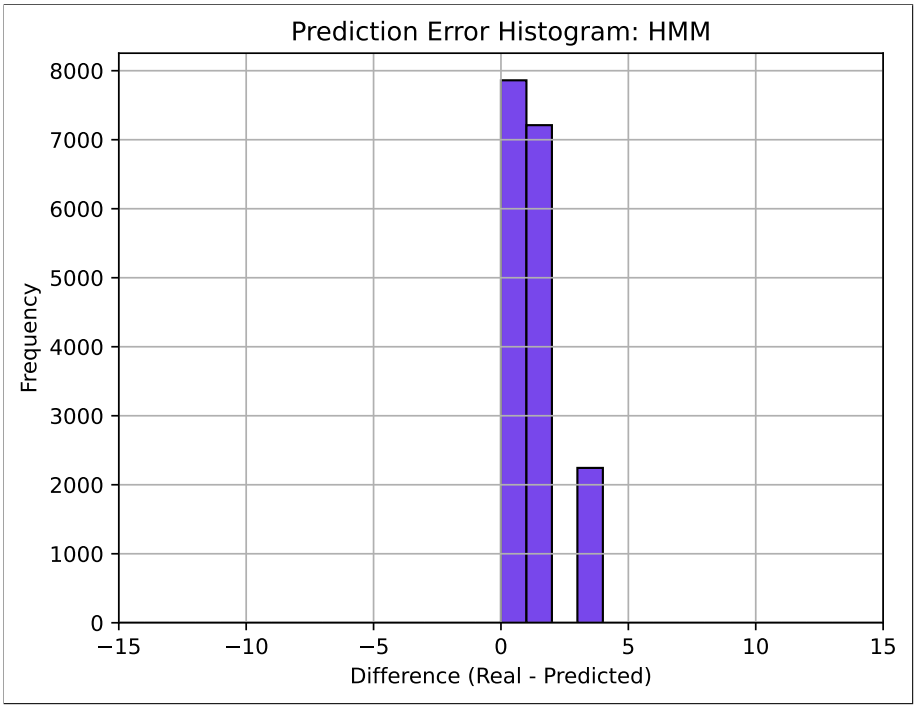

Figure 4.15: Full data set used on XY Axis coordinate data with HMM (histogram).

Results are presented in this chapter alongside an analysis of how the models behave when compared to their pairs. The collected information was measured in both a desktop laptop and in a *Raspberry Pi 5* environment, in order to study as well how the models prediction times are affected when running on an embedded system and not just the same training desktop laptop.

Table 4.1: Full Model Results - Running on Desktop

Model	**MSE (px)**	**Training (s)**	**Inference (s)**	**Index**
Ridge	56.913618	0.031249	0.000000	0.000419
LinearRegression	56.913618	0.048714	0.015623	0.000496
XGBoost	43.731809	0.468935	0.000000	0.000783
LightGBM	54.227856	0.397047	0.031245	0.000903
DecisionTree	**1.300243**	1.006892	0.000000	0.001033
KNN	30.845719	0.330785	0.650366	0.002966
Lasso	57.066043	5.207217	0.000000	0.005900
CatBoost	52.615412	8.736185	0.062494	0.009836
GradientBoosting	60.760937	17.840946	0.012994	0.019352
RandomForest	**2.314412**	69.899276	0.366791	0.075345
LSTM	70.017443	112.812474	1.234196	0.124524
HMM	66338.389293	12.136937	0.013607	0.512867
SVR	68.523102	472.309095	134.028406	1.000507

It can be noted in 4.1 that the model with the smallest RMSE, is the Decision tree, closely followed by the RFR, but all the other models have RMSEs 10 times greater or even more.

Regarding the *Training time*, most models were below the 15 seconds mark, but Gradient boosting, Random Forest Regressor, LSTM and SVR took significantly more time. When it comes to the *inference time*, Most models were again below the 2 seconds mark, but the SVR took a significantly big time, marking it already as *Not recommendable* for this user application.

In Figure 4.16, the results of Table 4.1 are represented. Please note that due to the difference between the *Performance Index* of HMM and SVR models, it makes

more sense to show the results in a logarithmic way, so the smaller index are actually comparable.

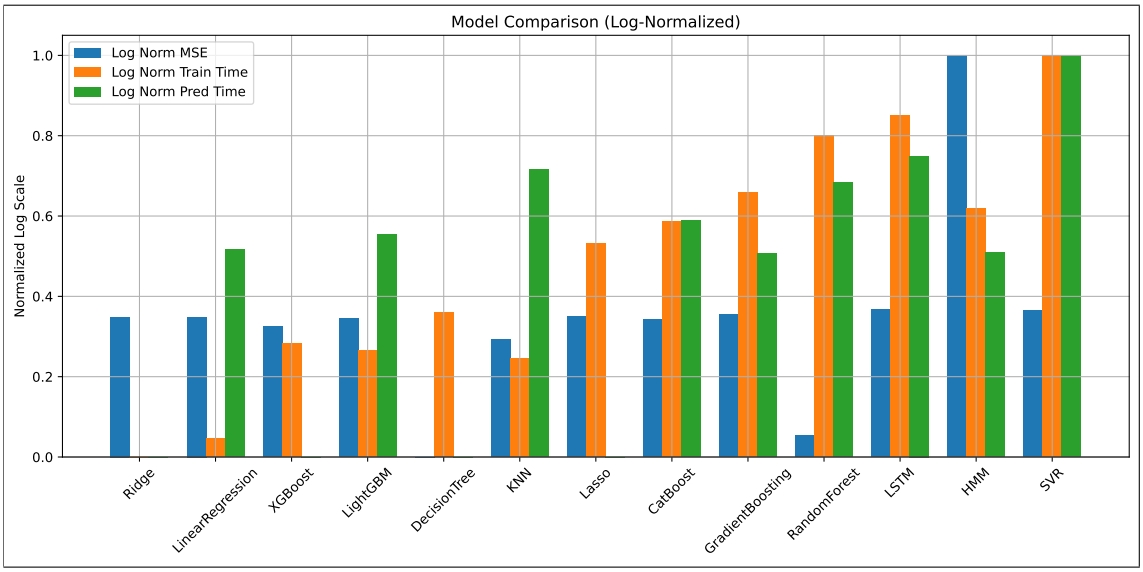

Figure 4.16: Comparison of models - Logarithmic Presentation - Running on Desktop.

Additionally, the file sizes of the resulting trained models are shown in Figure 4.17, note that the figure is represented using a logarithmic scale.

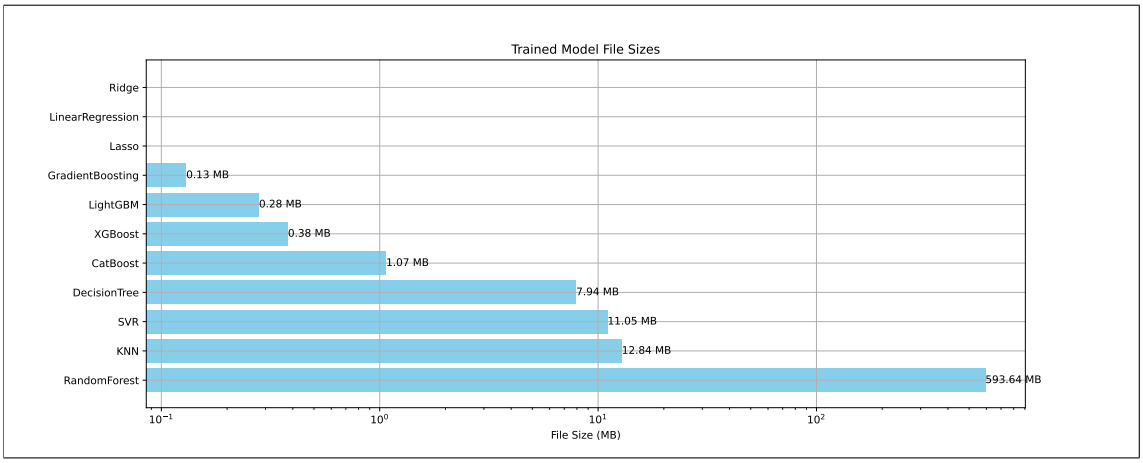

Figure 4.17: Generated model file sizes in logarithmic presentation.

After this point, the models have to be converted to *Tensor Flow Lite* [38], so they can run in embedded system and/or portable platforms such as *RaspberryPi5*.

4.3 Training and inference results on *RaspberryPi5*

Once the TFlite models have been run in the *RaspberryPi 5*, the measurements collected are presented in Table 4.2, where it can be seen that the RMSE values are now close together and the inference time in most cases is around 0.13 seconds.

Regarding CPU usage on the *RaspberryPi 5* when doing inference, it did not went over 30 percent in most cases, except for the *Decision tree*.

Then again, a *Performance index* was calculated by normalizing and adding up the collected metrics, used to sort the models in Table 4.2 keeping the smallest at the top.

Table 4.2: Tensor Flow lite Models - Running on *RPi 5* - CPU Usage

Model	**MSE (px)**	**Inference (s)**	**CPU (%)**	**Index**
LinearRegression	45.348389	0.135158	0.90	0.247625
RandomForest	40.124916	0.135082	25.90	0.291646
LightGBM	41.773418	0.135022	27.30	0.357018
GradientBoosting	41.712837	0.135355	25.90	0.379864
Lasso	40.252144	0.136082	25.90	0.411540
Ridge	42.141010	0.136117	25.90	0.483122
XGBoost	46.017742	0.134825	27.30	0.485946
SVR	39.496738	0.137714	27.30	0.586207
DecisionTree	44.221981	0.135635	35.70	0.595759
CatBoost	48.822899	0.135216	27.30	0.631394
KNN	46.450115	0.136158	25.90	0.641840

Additionally, the normalized metrics are presented in Figure 4.18, for a visual comparison of the inference results when running on a *RaspberryPi5*.

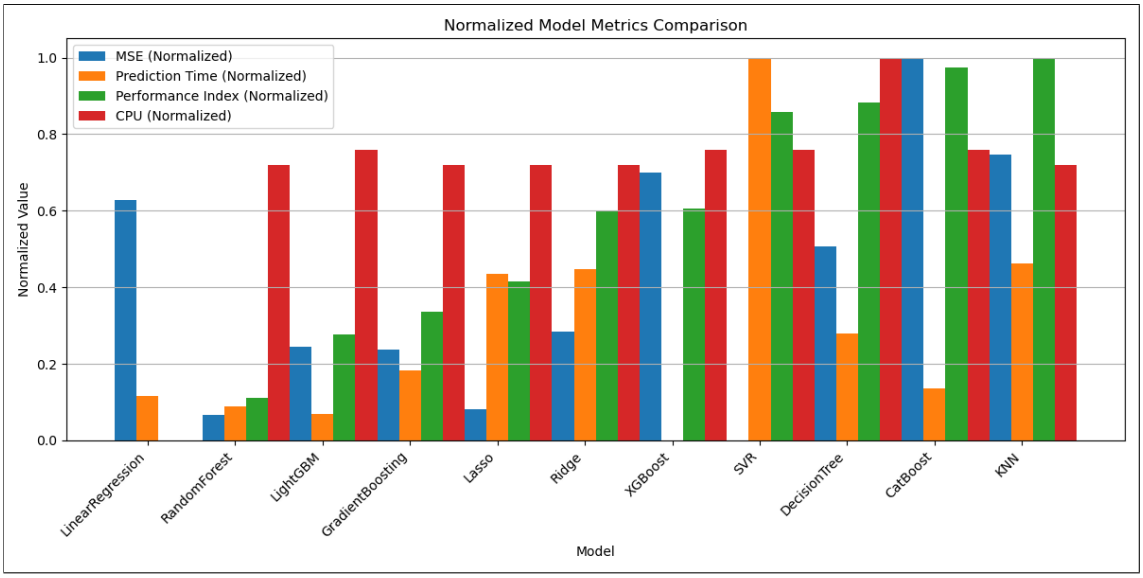

Figure 4.18: Normalized model metrics comparison running on *RaspberryPi5*.

In addition to measuring the performance of the machine learning models, the electrical current consumption usage of the Raspberry Pi 5 was also recorded to better understand how much energy the device consumes during model inference. To do this, a USB tester device, specifically the ***UNI-T UT658B***, was used to monitor both the voltage ($5.10[V]$) and electrical current going into the *Raspberry Pi*. These measurements were taken under two different conditions: when the device was *idle* (Turned on, but making no inferences) ($0.57[A]$), and when it was actively *running inference* with the trained machine learning models ($0.90[A]$). The results showed that there was a noticeable difference in current consumption between these two states. The change in electrical current, which can be calculated as:

$$0.90[A] - 0.57[A] = 0.33[A].$$

5 Conclusions

On these chapter, the final conclusions of the research developed in work is presented. After the analysis, design, and implementation stages discussed in the previous chapters, it is now possible to summarize some key findings, reflect on the objectives that were proposed at the beginning, and discuss how they were accomplished, and highlight the technical results that were achieved.

- The trained MLA is best when trained on the augmented data and tested over the original one rather than just using the original one (This had to be verified as it might not always be the case)

- The more data used to train the MLA, the more accurate it can be but the file size gets bigger, so a balance in between this traits must be studied per user case

- When running preliminary inference test on the desktop laptop, Decision Three MLA is the most accurate model, with just a slightly bigger training time, but this is almost negligible, not only because it is around one second only, but because the other top MLAs are not that far off, and it definitely compensates by having a prediction time close to zero seconds. (Table 4.1)

- Nevertheless, a plot twist occurs when models are converted to tensor flow lite and run in a portable system suchs a raspberry pi 5, Decision three has the highest CPU usage and is not among the ones with the less error (Table 4.2)

- In this presented portable system case of the *RaspberryPi5*, the SVR had the smallest error, with an inference time and a CPU usage not that far from its peers. So in this study case is the one that fulfils best the requirements, along wiht the RF, but if more portable platforms are tested in the future then a different MLA could be more suitable. Nevertheless, a methodology to test and verify this has been given in this document.

- The overall power consumption difference in the *RaspberryPi5* is not as big as it was expected, reaching only up to about 0.33 [A]

5.1 Future work

Even though this work mostly covers the main goals proposed, there are still several directions that could be explored in the future. Some limitations were found during the work, and there are ideas that could not be developed due to time or resource constraints. This section presents a list of suggestions for improvements, possible extensions, and new questions that came up during the research.

- **Explore the best balance between file size and model accuracy:** It was noted when training some models, that experimenting with different augmented data sets and different data set sizes the file size of the resulting models would change from a few megabytes to several hundreds. So, *optimizing* to find the best *file size against model performance* is a task to do depending on the embedded system that the microscope could have, the storage limitations and the required model performance. The goal would be to find a point where the file size is reduced enough to be practical, but the accuracy is still with in the required range.

- **Test the implementation on FPGA hardware:**

So far, the experiments were done using the a desktop laptop for training and preliminary evaluation, and a *RaspberryPi5* for embedded/portable system inference testing, but it would be very valuable to try this concept on an FPGA. These devices are efficient and can run models faster and with lower power, which is great for real-time computing. Future work could focus on adapting the model for FPGA, checking how many hardware resources it uses, and measuring how fast and efficient it is when doing predictions.

- **Study alternative models for image segmentation:** In this work, the segmentation model used to generate the binarized version of the images took approximately 2.4 seconds per image when running on the Raspberry Pi 5. This processing time may not be acceptable for real-time applications or systems that require higher throughput. Future research could explore lighter or more efficient segmentation models that maintain good quality while reducing inference time. Some options might include using smaller architectures like MobileNet-based variants, or experimenting with model optimization techniques specific to ARM architectures.

- **Benchmark other hardware platforms, such as Microchip MPUs:** While the Raspberry Pi 5 was used in this thesis due to its accessibility and general-purpose performance, it is not necessarily the most efficient platform for all embedded applications. Future work could involve benchmarking the same image analizys pipeline on alternative hardware platforms, such as Microchip's MPUs (for example the SAMA5 or MPFS series)[39], which are designed for low-power and real-time processing. Comparing performance metrics such as inference time, CPU usage, memory consumption, and power efficiency. This was attempted during this work but it could not be completed due to OpenCV and TFLite library constrains and compatibility on the Microchip SAMA5 series when using specific versions of *Buildroot*.

- **Evaluate implementation of these models in a real case scenario:** Use a microscope that has an XY stage and runs on an embedded system that is able to run the trained models. The system should be used to collect real-time data while the models are making predictions. This data should include how accurate the model is, how long it takes to make predictions, and how efficient it is in terms of resources used. After collecting this data, do a comparison between the models to see which one works better. Also, check if the images produced during this process are clear and good enough to be used for scientific analysis or research.

References

[1] F. Xing, Y. Xie, H. Su, F. Liu, and L. Yang, "Deep learning in microscopy image analysis: A survey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 10, pp. 4550–4568, 2018. DOI: 10.1109/TNNLS.2017.2766168.

[2] C. Tian, C. Yang, and S. L. Spencer, "Elliptrack: A global-local cell-tracking pipeline for 2d fluorescence time-lapse microscopy", *Cell Reports*, vol. 32, no. 5, p. 107 984, 2020, ISSN: 2211-1247. DOI: https://doi.org/10.1016/j.celrep.2020.107984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2211124720309694.

[3] F. Lavitt, D. J. Rijlaarsdam, D. van der Linden, E. Weglarz-Tomczak, and J. M. Tomczak, "Deep learning and transfer learning for automatic cell counting in microscope images of human cancer cell lines", *Applied Sciences*, vol. 11, no. 11, 2021, ISSN: 2076-3417. DOI: 10.3390/app11114912. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/4912.

[4] A. C. Y. Wu and S. A. Rifkin, "Aro: A machine learning approach to identifying single molecules and estimating classification error in fluorescence microscopy images", *BMC Bioinformatics*, vol. 16, p. 102, 2015. DOI: 10.1186/s12859-015-0534-z. [Online]. Available: https://doi.org/10.1186/s12859-015-0534-z.

[5] Y. Li, J. Di, K. Wang, S. Wang, and J. Zhao, "Classification of cell morphology with quantitative phase microscopy and machine learning", *Opt. Express*, vol. 28, no. 16, pp. 23 916–23 927, Aug. 2020. DOI: 10.1364/OE.397029. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-28-16-23916.

[6] M. Ali, V. Benfante, G. Basirinia, *et al.*, "Applications of artificial intelligence, deep learning, and machine learning to support the analysis of microscopic images of cells and tissues", *Journal of Imaging*, vol. 11, no. 2, 2025, ISSN: 2313-433X. DOI: 10.3390/jimaging11020059. [Online]. Available: https://www.mdpi.com/2313-433X/11/2/59.

[7] Aboa-Space-Research-Oy, *Mini fluorescence microscope (mfm)*, European Space Agency, 2022. [Online]. Available: https://nebula.esa.int/content/mini-fluorescence-microscope-mfm.

[8] H. C. Koydemir, Z. Gorocs, D. Tseng, *et al.*, "Rapid imaging, detection and quantification of giardia lamblia cysts using mobile-phone based fluorescent microscopy and machine learning", en, *Lab Chip*, vol. 15, no. 5, pp. 1284–1293, Mar. 2015.

[9] C. L. Cooke, F. Kong, A. Chaware, *et al.*, "Physics-enhanced machine learning for virtual fluorescence microscopy", 2020. eprint: 2004.04306 (eess.IV).

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation", in *Lecture Notes in Computer Science*, ser. Lecture notes in computer science, Cham: Springer International Publishing, 2015, pp. 234–241.

[11] A. Zargari, G. A. Lodewijk, N. Mashhadi, *et al.*, "DeepSea is an efficient deep-learning model for single-cell segmentation and tracking in time-lapse microscopy", en, *Cell Rep. Methods*, vol. 3, no. 6, p. 100 500, Jun. 2023.

[12] C. Qu, X. Che, Z. Cai, H. Li, and S. Wang, "Safety boundary in virtual reality: An approach based on user motion analysis and prediction", in *2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta)*, Haikou, China: IEEE, Dec. 2022, pp. 1942–1947.

[13] A. Mosavi, P. Ozturk, and K.-W. Chau, "Flood prediction using machine learning models: Literature review", en, *Water (Basel)*, vol. 10, no. 11, p. 1536, Oct. 2018.

[14] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines", en, *Ore Geol. Rev.*, vol. 71, pp. 804–818, Dec. 2015.

[15] I. S. Candanedo, E. H. Nieves, S. R. González, M. T. S. Martín, and A. G. Briones, "Retracted chapter: Machine learning predictive model for industry 4.0", in *Knowledge Management in Organizations*, L. Uden, B. Hadzima, and I.-H. Ting, Eds., Cham: Springer International Publishing, 2018, pp. 501–510.

[16] P. F. Smith, S. Ganesh, and P. Liu, "A comparison of random forest regression and multiple linear regression for prediction in neuroscience", en, *J. Neurosci. Methods*, vol. 220, no. 1, pp. 85–91, Oct. 2013.

[17] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines", en, *Ore Geol. Rev.*, vol. 71, pp. 804–818, Dec. 2015.

[18] *The cell image library*, `https://www.cellimagelibrary.org/home`, 2025.

[19] *Allen cell explorer*, `https://www.allencell.org/`, 2025.

[20] *The human protein atlas*, https://www.proteinatlas.org/.

[21] *Image data resource (idr)*, https://idr.openmicroscopy.org/, 2025.

[22] A. Zargari, G. A. Lodewijk, C. W. Neudorf, *et al.*, *Deepseas website*, 2021. [Online]. Available: https://deepseas.org/datasets/.

[23] *Figshare fluorescence*, https://figshare.com/, 2025.

[24] A. Zargari, G. A. Lodewijk, C. W. Neudorf, *et al.*, "Deepsea: An efficient deep learning model for automated cell segmentation and tracking", *bioRxiv*, 2021, Preprint. DOI: 10.1101/2021.03.10.434806. [Online]. Available: https://doi.org/10.1101/2021.03.10.434806.

[25] *Deepsea datasets*, https://deepseas.org/datasets/, 2025.

[26] Scikit-learn, *Linearregression*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html, 2025.

[27] Scikit-learn, *Ridge*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html, 2025.

[28] Scikit-learn, *Decisiontreeregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html, 2025.

[29] Scikit-learn, *Randomforestregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, 2025.

[30] Scikit-learn, *Gradientboostingregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html, 2025.

[31] Scikit-learn, *Svr*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html, 2025.

[32] Scikit-learn, *Kneighborsregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html, 2025.

[33] XGBoost, *Xgbregressor*, https://xgboost.readthedocs.io/en/latest/python/python_api.html, 2025.

[34] LightGBM, *Lgbmregressor*, https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html, 2025.

[35] CatBoost, *Catboostregressor*, https://catboost.ai/docs/en/concepts/python-reference_catboostregressor.html, 2025.

[36] TensorFlow, *Tf.keras.layers.lstm*, https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM, 2025.

[37] hmmlearn, *Hidden markov models in python*, https://hmmlearn.readthedocs.io/, 2025.

[38] Google AI Edge, *Litert: High-performance runtime for on-device ai*, 2024. [Online]. Available: https://ai.google.dev/edge/litert.

[39] Microchip Technology Inc., *Sama7 microprocessors (mpus)*, https://www.microchip.com/en-us/products/microprocessors/32-bit-mpus/sama7, n.d.