TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Javier Galindos Vicente 221138IAFM

FORECASTING CLOUD RESOURCE UTILIZATION USING MACHINE LEARNING AND COMPUTER VISION

Master Thesis

Supervisor
Thaleia Dimitra Doudali
Assistant Research Professor
IMDEA Software Institute
Mentor
Olutosin Ajibola Ademola
PhD Candidate
Tallinn University of Technology

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Javier Galindos Vicente 221138IAFM

PILVERESSURSSIDE KASUTAMISE ENNUSTAMINE MASINÕPPE JA TEHISNÄGEMISE ABIL

Magistritöö

Juhendaja
Thaleia Dimitra Doudali
Uurimisprofessori assistent
IMDEA Software Institute
Kaasjuhendaja
Olutosin Ajibola Ademola
Doktorant
Tallinna Tehnikaülikool

Tallinn 2022

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Javier Galindos Vicente
Date: $3^{rd} June, 2022$

Acknowledgments

Throughout the writing of this Master's thesis, I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Thaleia Dimitra Doudali, whose expertise was invaluable in formulating the research questions and project scope. Your dedicated guidance and insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would also like to thank the IMDEA Software Institute for providing a rich and collaborative environment where I can freely develop my ideas and grow as a professional. Nor can I forget my colleagues here, especially Aristotelis, who have provided me with help whenever I have needed it and constructive talks.

I would like to acknowledge my university mentor, Olutosin Ajibola Ademola, and the university, TalTech, for their valuable guidance throughout my studies. You provided me with the tools I needed to choose the right direction and successfully complete my master's. Additionally, I would like to thank Peeter Ellervee, for their unconditional help in all the matters I have needed.

Finally, I would like to thank my classmates and now friends, who have made this journey so much easier, more fruitful, and enjoyable.

Abstract

Cloud computing has revolutionized the access and use of computing hardware technologies at a massive scale. The cloud computing paradigm, despite its enormous success and numerous benefits, faces several obstacles. A major challenge is resource provisioning for computational tasks. There is a need to make accurate forecasts of resource utilization to achieve efficient resource management and cost efficiency in cloud environments. Cloud resource utilization traces are rather complex and random. Traditional time series forecasting methods are not able to capture the non-linearity and complexity of cloud resource utilization. In recent times, several machine learning approaches have attempted to solve this problem using more sophisticated models, but fail to provide highly accurate predictions in return for low training and inference times. Recent work in the financial domain shows how the use of an image representation of time series data, and relevant image-based methods can lead to more robust and effective forecasting. To this end, this thesis explores the use of images, computer vision and machine learning methods to forecast future resource utilization in cloud environments. An image-based prediction pipeline is proposed, that visualizes data in a sophisticated way, uses image-based machine learning methods to predict resource consumption, similar to those used in video frame prediction, and then decomposes the predicted images back to numeric predictions. Furthermore, this thesis includes an in-depth comparative analysis that shows how an image-based prediction pipeline can provide accurate forecasts for long windows of time in the future, as well as capture the short-term patterns and overall trends of the data. Most importantly, the proposed image-based machine learning method, typically used in video frame prediction, can accurately forecast resource utilization, even when the training and inference datasets exhibit completely different characteristics, something that is currently not possible using other image-based, non-image-based and traditional forecasting methods.

The thesis is in English and contains 111 pages of text, 9 chapters, 65 figures, and 20 tables.

List of abbreviations and terms

ANN Artificial Neural Network.

ARIMA Auto Regressive Integrated Moving Average.

BLSTM Bidirectional Long Short-Term Memory.

BRNN Bidirectional Recurrent Neural Networks.

CNN Convolutional Neural Network.

DTW Dynamic Time Warping.

EMA Exponential Moving Average.

GAF Gramian Angular Field.

GARCH Generalized AutoRegressive Conditional Heteroskedasticity.

GRU Gated Recurrent Units.

IoU Intersection-over-Union.

LRCN Long Recurrent Convolutional Network.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MASE Mean Absolute Scaled Error.

ML Machine Learning.

QoS Quality of Service.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

SIFT Scale-Invariant Feature Transform.

SLA Service Level Agreement.

SVM Support Vector Machine.

VM Virtual Machine.

Table of Contents

List of Figures

List of Tables

1. Introduction

Cloud computing has revolutionized the access and use of computing hardware technologies at a massive scale. It provides a customizable platform that allows users to configure their applications to acquire appropriate resources before execution over a virtualized computing environment, by deploying and using Virtual Machine (VM)s [1]. The wide spectrum of advancements in cloud computing has resulted in considerable growth in users of cloud technologies and the creation of specialized management systems for resource load balancing [2]. The cloud computing paradigm, despite its enormous success and numerous benefits, raises many challenges regarding resource, power and cost efficiency. In particular, efficient resource provisioning entails dynamic resource scaling (up and down) in response to present and future demands generated by the workloads executing over a cloud computing platform. To achieve efficient resource management and cost efficiency in cloud environments, accurate resource utilization predictions are necessary. To this end, there needs to be systems and mechanisms that capture and analyze workload characteristics and resource utilization behaviors, in order to build robust prediction models and forecast future resource utilization. In this way, cloud users can efficiently provision the distribution of the underlying hardware resources across the VMs.

Previous research has been carried out in cloud computing for resource utilization prediction to overcome over-provisioning and under-provisioning issues. Over-provisioning of resources consumes more energy and leads to high costs. However, under-provisioning induces Service Level Agreement (SLA) violation and Quality of Service (QoS) degradation [3]. To efficiently provision cloud resources, many traditional methods have been widely used. These methods include linear regression, moving averages, exponential smoothing, Auto Regressive Integrated Moving Average (ARIMA) models and modular regression models [4]. However, in recent years, the use of machine learning techniques in this field has received increased attention. Multiple works have explored the use of different machine learning algorithms to forecast cloud resource utilization. These include the use of Artificial Neural Network (ANN)s [5], Convolutional Neural Network (CNN)s [6], Recurrent Neural Network (RNN)s [7, 8] and Support Vector Machine (SVM) [9]. The increased intelligence of the machine learning models allows for more robust prediction models, especially for longer forecasting horizons compared to the aforementioned traditional methods. Being able to accurately forecast resource utilization for a longer period of

time is key to reducing energy consumption and optimizing resource allocation in cloud environments. Nonetheless, it is still very challenging to achieve high prediction accuracy in return for low training and inference times, especially for long forecasting horizons.

Many of the existing solutions in cloud resource management, use techniques that are traditionally used in forecasting generic time series data. In that domain, traditional methods include linear regression, exponential smoothing, ARIMA or Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) [10, 11, 12], as well as more recent machine learning solutions exploring the use of ANNs, CNNs and RNNs [13, 14, 15]. However, the way modern workloads utilize cloud resources over time, is very complex and typically does not exhibit a distinct pattern or seasonality [5], resembling a random distribution. To capture such complexity, recent work in forecasting time series of financial data explores the use of image-based machine learning methods [16, 17]. They show how visualizing the actual timeseries data as an image and leveraging a two dimensional representation of the data, is able to capture both temporal and spatial correlations of the data, which numeric approaches struggle to extract, resulting in more robust forecasting models.

1.1 Statement of Problem

To achieve efficient resource management in cloud environments, it is essential to build accurate resource utilization prediction models. However, the nature of cloud resource utilization traces may be very complex and random. The trends and patterns differ significantly between different workloads and deployments of virtual machines. Recent solutions leverage machine learning methods to deal with the complexity and non-linearity of the data and improve prediction accuracy. However, they have significant overheads and frequently fail to predict resource utilization for longer forecasting horizons or capture short term data patterns. Inspired by work in forecasting financial time series data using an image-based prediction model [17], this thesis explores the effectiveness of visualizing the cloud resource utilization across time as an image and using image-based machine learning models, similar to those used for video frame prediction, to improve upon efficient cloud resource provisioning.

1.2 Thesis Contributions

This thesis explores the effectiveness of using image-based machine learning methods to forecast cloud resource utilization. We deal with different visualization challenges and overheads, explore various image-based methods and evaluate how they compare with current state-of-the-art machine learning and traditional timeseries forecasting methods.

1. Methodology to create, process and decompose images, which visualize cloud resource utilization, to use for machine learning.

We create image representations of numeric cloud resource utilization traces from a real dataset. This dataset contains data from a datacenter with over a thousand virtual machines and is characterized in Chapter 5. We visualize the numeric time series data as lines in a two dimensional image, where the x-axis is the time and the y-axis is the value of the resource usage at each specific timestep. We explore various libraries to create the images and different image sizes, to decide which ones show a clear pattern in the images, and avoid information loss. Afterward, we structure the images into a sequence with overlap, to input them into the machine learning pipeline and leverage both the temporal and spatial information. Finally, we propose a methodology to decompose the images back to numeric data, to be used as the final prediction of resource utilization. (Chapter 5, 6)

2. Evaluation of various state-of-the-art image-based machine learning methods for the purpose of cloud resource utilization forecasting.

After we have decided how to best create the image representation of the numeric data, we then evaluate the effectiveness of using image-based machine learning models for the purpose of cloud resource utilization forecasting. We choose three state-of-art methods: Convolutional Autoencoder, Long Recurrent Convolutional Network (CNN + Long Short-Term Memory (LSTM)) and a video frame prediction model that uses ConvLSTM modules. We compare their effectiveness along various configurations of critical hyperparameters and input formats, such as the length of the forecasting history, forecasting horizon, and overlap of the input sequence of images. Then, we optimize the three methods and compare them in terms of prediction accuracy, using numeric, image-based (object detection) and time series related error metrics; along with training and inference times and model sizes. Our most important findings are that models whose input is an image and the output is also an image best capture the short-term patterns of the data while image-to-numeric models only capture the trend of the data. In addition, models that contain an LSTM cell inside their architecture are more robust under changes in image representation and critical hyperparameters, notwithstanding the fact that other models may obtain maximum performance. (Chapter 7)

3. Comparison of image-based methods against other machine learning and traditional forecasting methods for predicting cloud resource utilization.

Lastly, we compare the aforementioned image-based machine learning methods against other machine learning and traditional methods used for forecasting cloud resource uti-

lization (i.e., ARIMA, exponential smoothing, and LSTMs). We configure the different methods with the hyperparameters that yield the best performance, which we summarize in Chapter 4, and analyze in Chapters 7 and 8. We then evaluate the prediction accuracy for increasing forecasting horizons, since it is important and very challenging for cloud environments to make accurate predictions for long windows in the future [18] to optimize resource provisioning. Our findings show that the video frame prediction model has the capability to accurately forecast the short-term patterns and trends of the utilization patterns for long forecasting windows, when traditional, numeric and other image-based methods completely fail. Finally, we evaluate the prediction capabilities of the above models when using the trained models to perform inference over unseen data (workloads executing over different virtual machines). These data can potentially exhibit a completely different behavior and patterns across time. We use a clustering algorithm to group VM depending on their pattern of resource utilization across time (Chapter 5). We find that when two VMs belong to the same cluster and have similar patterns, the video-frame prediction accurately forecasts the trend and the short-term pattern of the data, while the Long Recurrent Convolutional Network (LRCN) only captures the overall trend. When the patterns of the two VMs are different, regardless if they are clustered together or not, only the video-frame prediction model is capable of making robust forecasts, while all other models (image-based, non-image-based and traditional) completely fail (Chapter 8).

In conclusion, this thesis proposes the use of images to capture the patterns of cloud resource utilization across workload execution time. We propose an image-based prediction pipeline, that visualizes data in a sophisticated way, uses image-based machine learning methods to predict resource consumption, similar to those used in video frame prediction, and then decomposes the predicted images back to numeric predictions. Our in-depth comparative analysis shows that an image-based prediction pipeline can provide accurate forecasts for long windows of time in the future, as well as capture the short-term patterns and overall trends of the data. Most importantly, we show that the proposed image-based machine learning method, used in video frame prediction, can accurately forecast resource utilization, even when the training and inference dataset exhibit completely different characteristics, something that is currently not possible using other image-based, non-image-based and traditional forecasting methods.

1.3 Document Organization

The remainder of this thesis document is organized as follows. The different methods utilized across this thesis are explained in Chapter 2. Chapter 3 summarizes the state-of-the-art of the related work of cloud resource utilization forecasting and time series forecasting. Chapter 4 describes the different methods, frameworks, and resources used.

In Chapter 5, the dataset is presented and analyzed, and a clustering algorithm is proposed. The different methods to process the data for the machine learning pipeline are presented in Chapter 6. Chapter 7 analyzes the effectiveness of the different image-based methods. The image-based models are compared with traditional methods and numeric machine learning models in Chapter 8. Finally, Chapter 9 concludes the most important findings of this thesis and gives an outlook on future directions.

2. Background

This chapter gives an overview of the different methods and algorithms explored in this thesis. First, the models utilized for the baseline in the domain of time series forecasting. Later, the computer vision and machine learning algorithms used are described.

2.1 Traditional Forecasting Methods

Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data [11].

Exponential smoothing

Exponentially smoothing forecasts are weighted averages of previous observations, with the weights decaying exponentially as the observations get older. In other words, the larger the related weight, the more recent the observation. This framework produces accurate forecasts fast and for a wide range of time series, which is a significant benefit for industrial applications [11].

Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations come from further in the past — the smallest weights are associated with the oldest observations:

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots \tag{2.1}$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter. The rate at which the weights decrease is controlled by the parameter α.

The weights connected to the observations decrease exponentially as we travel back in time for any α between 0 and 1, hence the name "exponential smoothing." When α is tiny (i.e., near to 0), observations from the remote past are given more weight. If α is large (i.e.,

near to 1), the more recent observations are given more weight.

ARIMA

ARIMA is the acronym of Auto Regressive Integrated Moving Average. The Auto Regressive model forecasts are based on a linear combination of past values of the variable. The Moving Average model forecasts based on a linear combination of past forecast errors. Basically, ARIMA models combine these two approaches. Since they require the time series to be stationary in mean and variance, differencing (Integrating) the time series may be a necessary step (i.e. considering the time series of the differences instead of the original one). A series should be differentiated by the number of times necessary to be stationary in mean. To make a series stationary in variance, box-cox or power transformations may be used [11].

In an autoregression model, we use a linear combination of the variable's historical values to forecast the variable of interest. The word autoregression denotes that the variable is being regressed against itself.

Thus, an autoregressive model of order p can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \tag{2.2}$$

where ε_t is white noise. This is like a multiple regression but with lagged values of y_t as predictors. This is referred as an $AR(p)$ model, an autoregressive model of order p.

Instead of using past values of forecast variables in regression, moving average models use past forecast errors in regression-like models:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \tag{2.3}$$

where ε_t is white noise. This is referred as an $MA(1)$ model, a moving average model of order q.

A non-seasonal ARIMA model is created by combining differencing with autoregression and a moving average model. ARIMA stands for AutoRegressive Integrated Moving Average (integration is the inverse of differencing in this context). The entire model can be written as follows:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{2.4}$$

where y'_t is the differenced series (it may have been differenced more than once). This is referred as $ARIMA(p, d, q)$ where p is the order of the autoregressive part, d is the degree of first differencing involved and q is the order of the moving average part.

2.2 Recurrent Neural Networks

A RNN is a form of artificial neural network that is designed to work with time series or sequence data. Ordinary feedforward neural networks are designed to handle only data items that are not related to each other. However, if we have data in a sequence where one data point is dependent on the preceding data point, we must change the neural network to account for these dependencies. RNNs feature a concept of memory, which allows them to store the states or information of prior inputs in order to construct the sequence's next output [19].

There are different types of RNN: **one-to-one** when there is only one input and one output, like traditional feed-forward networks; **one-to-many** where a single input can produce multiple outputs; **many-to-one** where many inputs from different time steps produce a single output; and **many-to-many** where many inputs produce many outputs.

There are different variations of RNNs that are being applied practically in machine learning problems to solve the problems of simplistic RNN (e.g., vanishing gradients): **Bidirectional Recurrent Neural Networks (BRNN)**, **Gated Recurrent Units (GRU)** and **Long Short Term Memory Networks (LSTM)**. LSTMS are the most widely used in the machine learning domain [19] and thus are utilized in this work.

Long Short Term Memory Networks LSTM is a special kind of RNN, capable of learning long-term dependencies. It is capable of handling the vanishing gradient problem faced by RNN.

The LSTM consists of three layers: the "forget gate layer", the "input gate layer" and the "output gate layer". The first part ("forget gate layer") chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part ("input gate layer"), the cell tries to learn new information from the input to this cell. Finally, in the third part ("output gate layer"), the cell passes the updated information from the current timestamp to the next timestamp. The LSTM also has a hidden state where h_{t-1} represents the hidden state of the previous timestamp

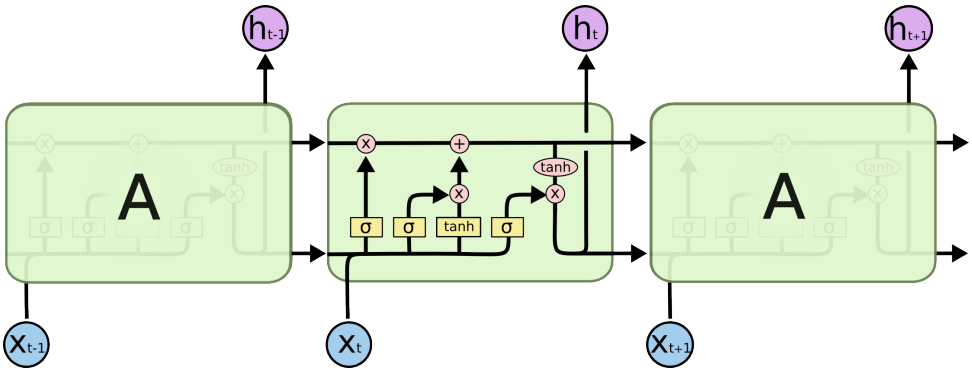


Figure 1. Representation of three LSTM cells. Source: Colah.github [20].

and h_t is the hidden state of the current timestamp. In addition, LSTM also has a cell state represented by $C(t-1)$ and $C(t)$ for the previous and current timestamps, respectively. The structure is presented in Fig. 1

2.3 Image-based Machine Learning Models

Deep Learning architectures have been applied to fields including computer vision, speech recognition, natural language processing, machine translation and bioinformatics, among others, where they have produced results comparable to and in some cases surpass human expert performance [21, 22]. These algorithms are bio-inspired by how human brain functions. DNN is a machine learning technique that allows computers to learn complex mapping between the input and output layers, then use these mappings learned on new dataset to make new predictions. In this thesis, we have utilized different deep learning models that are briefly explained in this section.

Convolutional Autoencoder

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function $h = f(x)$ and a decoder that produces a reconstruction $r = g(h)$. [19]

In this thesis, the convolutional autoencoder is used to produce a visual forecast image with the continuation of an input time series image, by learning an undercomplete mapping $g \circ f$,

$$\hat{y} = g(f(x)) \tag{2.5}$$

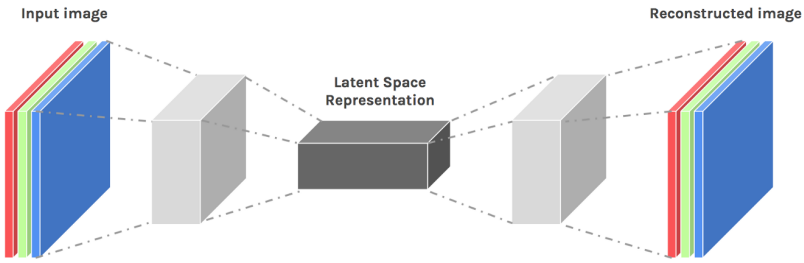


Figure 2. Example of a Convolutional Autoencoder. Source: AI Plainenglish [23].

where the encoder network $f(\cdot)$ learns meaningful patterns and projects the input image x into an embedding vector, and the decoder network $g(\cdot)$ reconstructs the forecast image from the embedding vector. [16]

The structure of a simple convolutional autoencoder is presented in Fig. 2. Inside the architecture, the encoder is usually formed by a convolutional layer followed by relu activation functions and max pooling layer. The output of this layer is flattened and input to a fully connected layer to connect to the latent space. The decoder is a mirrored version of the encoder network.

Video Frame Prediction Methods

The Convolutional LSTM architectures bring together time series processing and computer vision by introducing a convolutional recurrent cell in an LSTM layer.

The Convolutional LSTM or ConvLSTM is a sort of recurrent neural network that has convolutional structures in both the input-to-state and state-to-state transitions for spatio-temporal prediction. The ConvLSTM uses the inputs and past states of its local neighbors to predict the future state of a cell in the grid. Using a convolution operator in the state-to-state and input-to-state transitions is a simple way to accomplish this (observe Fig. 3) [24]. In other words, it is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations.

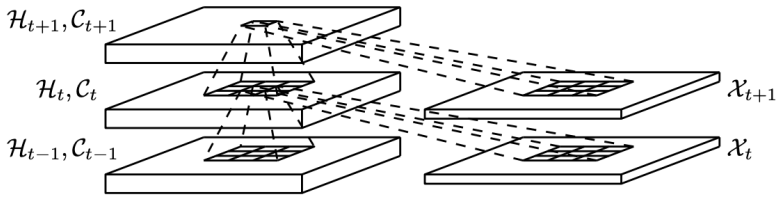


Figure 3. Inner structure of ConvLSTM. Source: Shi et al. [24].

The key equations of ConvLSTM are shown below, where $*$ denotes the convolution operator and $\odot$ the Hadamard product [24]:

$$i_t = \sigma\left(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \odot \mathcal{C}_{t-1} + b_i\right) \tag{2.6}$$

$$f_t = \sigma\left(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \odot \mathcal{C}_{t-1} + b_f\right) \tag{2.7}$$

$$\mathcal{C}_t = f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh\left(W_{xc} * X_t + W_{hc} * \mathcal{H}_{t-1} + b_c\right) \tag{2.8}$$

$$o_t = \sigma\left(W_{xo} * X_t + W_{ho} * \mathcal{H}_{t_1} + W_{co} \odot \mathcal{C}_t + b_o\right) \tag{2.9}$$

$$\mathcal{H}_t = o_t \odot \tanh\left(C_t\right) \tag{2.10}$$

The ConvLSTM layer could be stacked and combined with ReLu activation and max-pooling layers. Thus, Autoencoender-based models similar to those presented in the previous subsections could be built. Specifically, instead of using a convolutional layer, a ConvLSTM layer is utilized. With this approach, the model captures both spatial and sequential information.

Long Recurrent Convolutional Networks (LRCN)

Long Recurrent Convolutional Network (LRCN) are in general terms a combination of CNN and LSTM. The main idea is to use a combination of CNNs to learn visual features from video frames and LSTMs to transform a sequence of image embeddings into a class label, sentence, probabilities or the output needed. Thus, the raw visual input is processed with a CNN, whose output is fed into a stack of recurrent sequence models. The structure is presented in Fig. 4.

Whether using ConvLSTM, spatial and sequential information is processed simultaneously. On the other hand, utilizing LRCN the spatial is processed first, and the output of this part of the model is processed by the LSTMs.

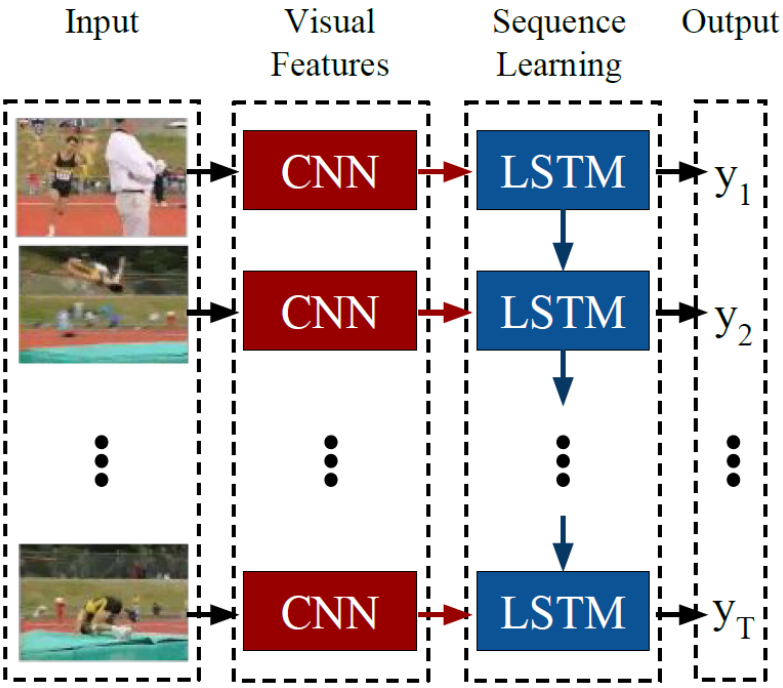


Figure 4. LRCN architecture. Source: Donahue et al. [25]

3. Related Work

3.1 Forecasting Cloud Resource Utilization

In this section, previous research work on forecasting cloud resource utilization is summarized. Previous publications could be grouped into traditional solutions and machine learning-based solutions.

3.1.1 Traditional Solutions

In the current literature, the prediction of resource usage required to refine the provisioning strategy is often done using traditional methods. These methods include linear regression, moving averages, exponential smoothing, ARIMA and Seasonal ARIMA models and modular regression models [4].

Sarikka et al. [26] compare the classical time series forecasting model with a machine learning approach such as LSTMs. They show how it is very important to analyze the nature of the data when using traditional methods. The classical time series models outperform the machine learning model in most of the analyzed scenarios, especially when the data is nonstationary. However, when the data are stationary and long-term, the machine learning model outperforms the classical models.

3.1.2 Machine Learning-based Solutions

In recent years, the use of machine learning techniques in research has received increased attention. One of the most extensively utilized methods in machine learning is neural networks. The ability to learn complex data relations and make accurate predictions is a significant benefit of employing neural networks. Multiple works have explored the use of machine learning to forecast cloud resource utilization as presented below, grouped by the type of machine learning method.

Artificial Neural Networks (ANNs)

Borkowski et al. [5] propose a machine learning method to build predictions about cloud

resource utilization from historical data. The created model is based on ANN. They used a real-world data set to evaluate their approach and compared it with a simple linear regression approach. In the median case, their model predicted the duration of cloud tasks with 20 % less prediction error compared to the baseline. In the best 5 % cases, 89 % less prediction error was achieved compared to the baseline.

Convolutional Neural Networks (CNNs)

Ouhame et al. [6] propose another algorithm. It targets multi-variate resource utilization prediction in cloud data centers. The resources include CPU, memory, and network bandwidth. The algorithm uses a CNN and LSTM models for resource utilization prediction. Initially, the vector autoregression method is used to filter the linear interdependencies between the multi-variate data. In the next step, CNN and LSTM are used for prediction. The proposed model is evaluated with experimental results and comparative results in terms of accuracy are presented.

Long-Short Term Memory (LSTM)

Gupta et al. [7] propose online learning of resource usage prediction models using gradient descent and Levenberg-Marquardt methods. They compare their approach that uses Bidirectional Long Short-Term Memory (BLSTM) with ARIMA, and observe that BLSTM outperforms ARIMA. To address the challenge of adapting a large number of parameters in BLSTM, sparse BLSTM is proposed. The adaptation time required for sparse and dense models is compared and it is seen that, due to the sparseness, the real-time adaptations are faster by 50-60% in the pruned model.

Yadav et al. [8] implement adaptive resource provisioning with an auto-scaling approach. The auto-scaling approach proposed in this work uses an LSTM model to estimate the future processing load of a web server, which is found out through historical data observations. They compare their approach with different ARIMA models and SVM. The results showed that the LSTM model performs better in terms of prediction accuracy than the SVM model and other ARIMA models.

Support Vector Machines (SVM)

Bankole et al. [9] explore three forecasting models using linear regression, neural network, and support vector regression to forecast cloud resource usage. Specifically, they forecast CPU utilization and two SLA metrics. The results show that the Support Vector Regression (SVR) model displayed superior prediction accuracy over both the Neural Network and

Linear Regression in a 9 to 12 minute window.

3.2 Forecasting Time Series

Time series forecasting is the task of making predictions of future values based on historical data. It entails developing models based on previous data and applying them to make observations and guide future strategic decisions. Some of the applications of time series forecasting include stock price prediction, weather forecasting, business planning, and resource utilization forecasting, among others.

Traditionally, statistical models such as linear regression, exponential smoothing, ARIMA or GARCH [10, 11, 12] have been widely used in time-series forecasting problems. In the past few years, new approaches have emerged as ensemble methods, tree-based methods, neural network auto-regressive models, and RNN [10] . We next describe related works, differentiating the ones that use image representations of the input data.

3.2.1 Machine Learning-based Solutions

In many machine learning problems, deep learning techniques have recently outperformed classical models. Deep neural networks have been used successfully to solve time series forecasting problems, which is an important topic in data mining. Given their ability to automatically understand the temporal connections found in time series, they have shown to be an effective solution. Unlike classical statistical-based models that can only model linear relationships in data, deep neural networks have shown a great potential to map complex non-linear feature interactions [27].

Lara-Benitez et al. [13] perform a thorough analysis of the different state-of-the-art models for time series forecasting. They use seven popular architectures: Multilayer Perceptron (MLP), Elman Recurrent Neural Network (ERNN), Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU), Echo State Network (ESN), Convolutional Neural Network (CNN) and Temporal Convolutional Network (TCN). They evaluate the performance of these models, in terms of accuracy and efficiency, over 12 different forecasting problems with more than 50000 time series in total. The main results point out that LSTM obtained the best results followed by GRU. However, CNN outperforms them in the mean and standard deviation of the error. This indicated that convolutional architectures are easier to parameterize than the recurrent models.

Salinas et al. [14] proposed a tool called DeepAR to make time series forecasts. DeepAR

is a probabilistic forecasting method that involves training an auto-regressive recurrent network model on a large number of related time series. They show how using deep learning techniques to forecasting can overcome many of the problems faced by traditional approaches to the problem. They demonstrate accuracy increases of roughly 15% compared to state-of-the-art approaches through extensive empirical evaluation on multiple real-world forecasting data sets.

Bandara et al. [15] propose a method to forecast across time series using LSTM trained on groups of similar series. These groups are formed using clustering algorithms. Methods to develop global models across such time series databases have been introduced to utilize the commonalities across multiple time series. However, in the presence of different time series, the accuracy of such a model may degrade, necessitating the inclusion of a concept of time series similarity. They suggested a forecasting framework that takes advantage of cross-series information in a set of time series by creating different models for subgroups of time series using an algorithmic clustering process. Their method is very competitive against models of different competitions and clearly outperforms the simplistic LSTM approach.

3.2.2 Image-based Solutions

Recently, new approaches have emerged that leverage image representation of time series. When displaying the underlying data in 2D graphics, visualizations convey spatial structure information [28] that is not present in the original data. Human eyes are skilled in capturing spatial structure or patterns in 2D images, which can aid in making better judgments or predictions. CNNs [29] have been proven to have the ability to extract characteristics of local spatial regions, allowing computers to recognize spatial patterns such as those in object identification and recognition tasks, thanks to advances in deep learning and computer vision.

Cohen et al. [30] suggest creating an image representation of a time series of the stock market to make more accurate classifications. They examine the value in transforming numerical time-series analysis to that of image classification in the financial domain. They train over a dozen machine learning classification models using the images, and discover that when the data is visually represented, the algorithms quickly retrieve the intricate, multiscale label-generating rules.

Sood et al. [16] propose a method for time series forecasting using images. They transform the numeric data into an image representation and use a convolutional autoencoder to make predictions. They validate their methodology in different synthetic and real datasets and

show how the image-driven approach outperforms numerical autoencoders and traditional methods such as ARIMA. The input of the model is an image, but the output is also an image, so finally they map the image back to numeric values.

Barra et al. [31] propose using Gramian Angular Field (GAF)s to train CNN models to build time series forecasting models in the financial domain. They claim that their method outperforms common strategies carried out in the stock market domain such as Buy & Hold (B&H). Their problem is a classification task; thus, the input is an image, but the output is a label.

Li et al. [32] also exploit an image representation of time series to make forecasts. In this case, the time series is transformed into an image using recurrence plots. They extract features from the image representation using Scale-Invariant Feature Transform (SIFT) or CNN. Later, they trained a different time series forecasting model (both classical and machine learning-based) and performed model averaging to make predictions. They evaluated their methods in two time-series competition datasets and showed that their models performed similarly to the top-ranked benchmarks in the competitions.

Zeng et al. [33] explored a method that leverages image representation of time series and video-frame prediction models to time-series forecasting problems. They used different stock market time series to generate an image and trained the video-frame prediction model. Later, they map back the output of the model (image) to numeric values. They evaluated their methods in nine financial assets traded in US stock markets and showed that their model outperforms baselines such as ARIMA and Prophet.

4. Experimental Methodology

In this chapter, the methodology used during this thesis is presented. The flow chart of the methodology is presented in Fig. 5. The first step is to understand the problem that we are facing so that we can find the appropriate dataset to solve the problem. Afterward, we explore and analyze the data (Chapter 5). Later, the data need to be processed to unleash their full potential for the models (Chapter 6). The next step is to find the most adequate models and optimize them (Chapter 7). The architecture and hyperparameters of the different models are described later in this chapter. Lastly, the models are compared and the efficiency of each model is evaluated (Chapter 8). The methods for evaluating the different models are also presented later in this chapter.

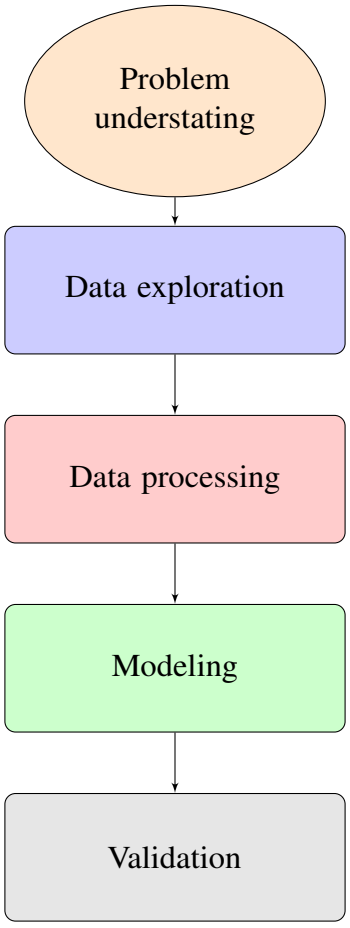


Figure 5. Flowchart of the experimental methodology.

4.1 Codebase and Libraries

The code of this thesis has been developed in Python3. The main libraries used are `Pandas`, `numpy`, `Scikit-learn`, `OpenCV`, `matplotlib`, `seaborn`, and `TensorFlow`.

The code of the thesis can be found in Github[1] and it is organized as follows: The folder DataExploration, contains all the code related to the data exploration and pre-processing and the clustering. The folder Modeling includes the corresponding code for the different models (i.e., Baseline, non-image-based machine learning and image-based machine learning) and the code for automating the experiments. The folder Figures has all the figures generated over the experiments performed during this thesis.

The code is developed locally. However, the experiments are run in a Linux server with native hardware. The code synchronization is performed using GitHub. The communication with the remote server is done through the ssh protocol. To automate the experiments a Python scripts with different arguments for the hyper-parameters have been created. Finally, to run many experiments, a Bash script is created to run multiple Python scripts.

For the training and deployment of the machine learning models, the chosen framework has been `TensorFlow` with the `Keras` API. TensorFlow is an open-sourced end-to-end platform, a library for multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Both provide high-level APIs that can be easily used to build and train models.

For all of the experiments performed during this thesis, some considerations have been taken into account. The training progress is tracked using TensorBoard. TensorBoard is the TensorFlow's visualization toolkit. It provides the visualization and tooling needed for machine learning experimentation. It has features such as tracking and visualizing metrics such as loss and accuracy, visualizing the model graph, viewing histograms of weights, biases, or other tensors as they change over time, etc.

4.2 Hardware Testbed

The experiments were conducted on a Linux machine with Ubuntu and the following hardware:

[1] `https://github.com/JavierGalindos/Forecasting-Cloud-Resource-Utilization-Using-Machine-Learning-and-Computer-Vision`

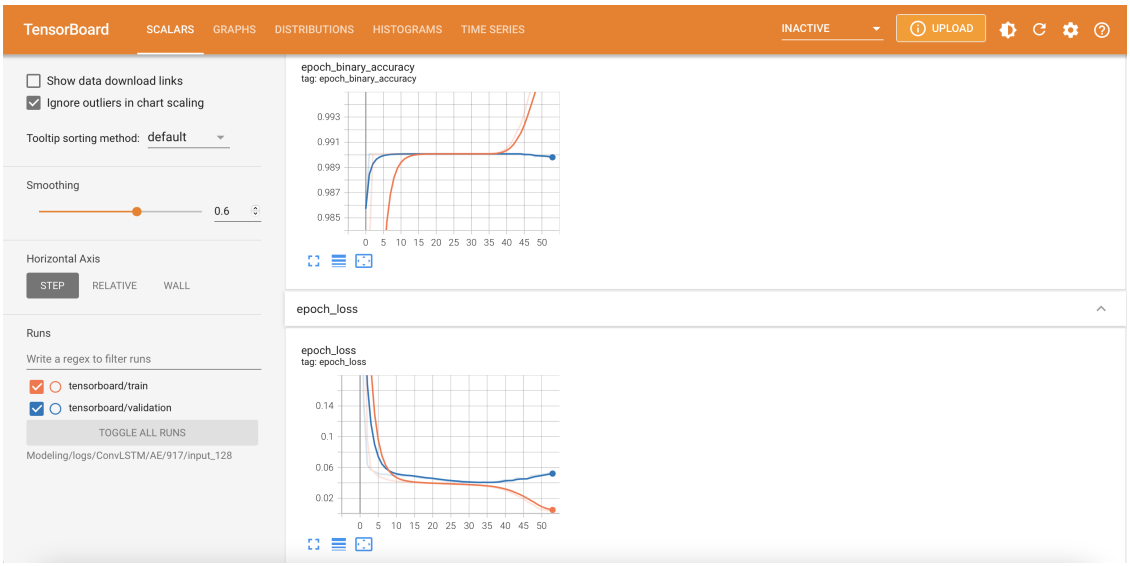


Figure 6. Example of TensorBoard.

- Intel(R) Core(TM) i5-8600 CPU @ 3.10GHz with 6 cores
- 16 GB RAM
- NVIDIA GeForce RTX 3090 GPU

To speed up the deep learning model training process, CUDA and cuDNN are installed.

First, the experiments were run on a different Linux machine with high-performance CPUs and professional hardware. However, high-end CPUs are significantly slower than GPUs in training deep learning models. Thus, it was needed to prepare a new Linux machine with GPU.

4.3 Evaluation Metrics

To avoid biased results, every metric presented in this thesis is computed over the test set, which is a complete unseen data subset for the model. Using the right metrics is key in every Data Science project. Having enough information about each model and each version of the model is essential to properly evaluate models and make comparisons between them. In this work, to analyze the accuracy of forecast predictions from each technique, we use a variety of metrics. Some of these measures are often used in the time series forecasting domain, while others are taken from the broader field of machine learning and applied to this problem. Furthermore, recent work [16], argues that numeric metrics may not capture all the information about time series forecasting problems and could be misleading in certain scenarios. Thus, additional image-based metrics are also examined. The metrics examined are the following:

- Mean Absolute Error (MAE)
- Root Mean Square Error (RMSE)
- Mean Absolute Percentage Error (MAPE)
- Mean Absolute Scaled Error (MASE)
- Dynamic Time Warping (DTW)
- Intersection-over-Union (IoU)
- Training time
- Inference time
- Model size

Prediction Accuracy Metrics

A selection of numeric errors is traced for time series. For every numeric error presented in this thesis, the smaller the error, the better the performance of the model.

Mean Absolute Error (MAE)

The MAE is a scale-dependent metric easy to calculate and interpret. It is computed as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.1}$$

where $\hat{y}_i$ is the forecast, y_i the corresponding observed ground truth, and n is the length of the time series.

It is easy to interpret, as it is on the same scale as the data. It does not penalize outliers. It is only suitable to compare models on the same data. To compare a model for different datasets, it is not suitable, as it is scale-dependent.

Root Mean Square Error (RMSE)

The RMSE is also an scale-dependent metric. It is computed as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.2}$$

where $\hat{y}_i$ is the forecast, y_i the corresponding observed ground truth, and n is the length of the time series.

Due to its square, it places more attention on outliers. Penalizes large errors compared to MAE. The error is also on the same scale as the data.

Mean Absolute Percentage Error (MAPE)

The MAPE is one of the most popular used error metrics in time series forecasting. It is a percentage error metric and is calculated as:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i} \cdot 100| \tag{4.3}$$

where $\hat{y}_i$ is the forecast, y_i the corresponding observed ground truth, and n is the length of the time series.

The main advantages are scale-independency and interpretability. It could be used to compare the output of different datasets with different scales.

The main disadvantage is that it generates infinite or undefined values for zero or close to zero values. In addition, it is asymmetric and places a hazier penalty on negative errors than on positive errors.

Mean Absolute Scaled Error (MASE)

The MASE is a scale-free metric. Removes the scale of the data by comparing the forecasts with those obtained from some benchmark (naive) method in the training set. The naive method is a random walk (forecast the last data point as future data). It is calculated as:

$$MASE = \frac{MAE}{MAE_{in-sample,naive}} \tag{4.4}$$

Errors less than 1 imply that the forecast performs better than the naive one-step method, with lower values indicating better predictions. As this error compares performance on different data subsets, whether the test subset is not a representative sample of the training subset, the metric could be inconsistent.

Dynamic Time Warping (DTW)

DTW is an algorithm to find an optimal alignment between two given (time-dependent)

sequences under certain restrictions [34]. DTW stretches the series along the time axis in a dynamic way over different portions to enable more effective matching. Unlike L_p metric (e.g., Euclidean distance), it allows many-to-one mappings. In turn, allows the artificial creation of two equal length series. (see Fig. 7) [35].

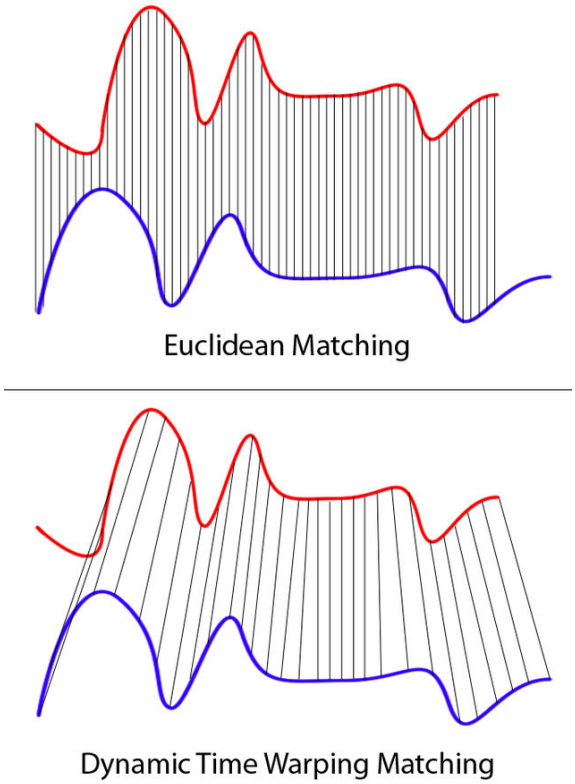


Figure 7. Comparison of the Euclidean distance and the Dynamic Time Warping (DTW) Matching. Source: Wikimedia [36]

Mathematically, DTW is defined as follows:

$$DTW(i,j) = distance(x_i, y_j) + min \begin{cases} DTW(i, j-1) & repeat\, x_i \\ DTW(i-1, j) & repeat\, y_j \\ DTW(i-1, j-1) & repeat\, neither \end{cases} \tag{4.5}$$

The lower DTW, the closer the ground truth series is to the forecast series and thus, higher the performance of the model. This metric is capable of reflecting when the model is learning the shape of the data and not only the trend of it.

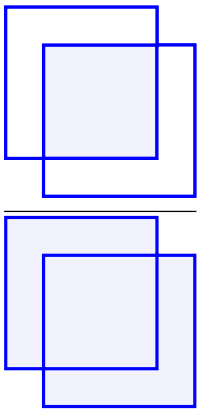
Intersection over Union (IoU)

In addition to using traditional forecast error metrics, we can measure the similarity between the predicted image and the ground truth image in our setting to evaluate model

performance.

The IoU metric is a common metric in object-detection problems [37, 38]. In this thesis, we extend the metric to measure forecast accuracy, similarly to the work done in [16]. It is defined as:

$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\quad}{\quad} \tag{4.6}$$



It ranges from 0.0 to 1.0, and higher values indicate more accurate forecasts.

The IoU is computed for each corresponding column in the ground truth and predicted image. This is done by obtaining the 1D bounding boxes of nonzero pixels for each column and then calculating the IoU of corresponding columns from the ground-truth and predicted image. Whether the algorithm does not produce an image, the latter is generated using the numpy technique presented in Section 6.1.1.

While the numeric metrics reward the model that learns the trend of the data, the IoU is better able to reflect how a model learns the short-term pattern of the data.

Other Evaluation Parameters

Lastly, it is also worthwhile to track some parameters of the model, such as the training time, inference time, or the model size. When the model goes to production, there could be some time or hardware constraints that prevent the implementation of certain models, although those outperform others in accuracy.

The **training time** is the time the model takes to train, the number of epochs may vary from one experiment to another due to the early stopping.

The **inference time** is the time it takes for the model to make the prediction for the entire

test subset. Whether a model needs to make an update towards making the next prediction, this time is also considered in the inference time.

The **model size** is the weight of the model in MB after saving it locally. Traditional models such as ARIMA or exponential smoothing do not have a certain weight, and thus this parameter does not apply to them and they are reflected as 'NaN'.

4.4 Neural Network Architecture and Hyperparameters

In this section, the details of the machine learning models utilized during this thesis are presented.

The selected optimizer is Adam [39]. The start learning rate is 0.001. Reduction of the learning rate on plateau is implemented. Whether the validation loss does not decay in 15 consecutive epochs, the learning rate is reduced by a factor of 10. Furthermore, to avoid overfitting and unnecessary long training time, early stopping is also implemented. The number of epochs is always 100 but, whether the validation loss does not decay in 20 consecutive epochs, the model stops training. Lastly, the best model of the training process is saved in the format *.hdf5*.

For the sake of reproducibility, a seed is set for every experiment performed. The seed has no effect on the train/val/test split due to the nature of time series. Additionally, the final results may vary due to the stochastic nature of machine learning models.

Convolutional Autoencoder

The autoencoder model is a simplistic convolutional autoencoder whose input is an image and the output is also an image. The encoder network learns the meaningful patterns and projects the input image into a latent space. Later, the decoder network reconstructs the image from the latent space. The model architecture is composed of two 2D convolutional layers with a kernel size of 5×5, stride 2, and padding 2. Every layer is followed by ReLU activation function and batch normalization layer. After each convolutional layer, the size of the image is reduced by 2. The first convolutional layer contains 128 filters and the second one 267 filters. The decoder network is a mirrored version of the encoder. The detailed architecture developed in TensorFlow is presented in Fig. 8.

Long Recurrent Convolutional Network (LRCN)

LRCN [25] are a combination of a CNN and an LSTM. The image is filtered through the

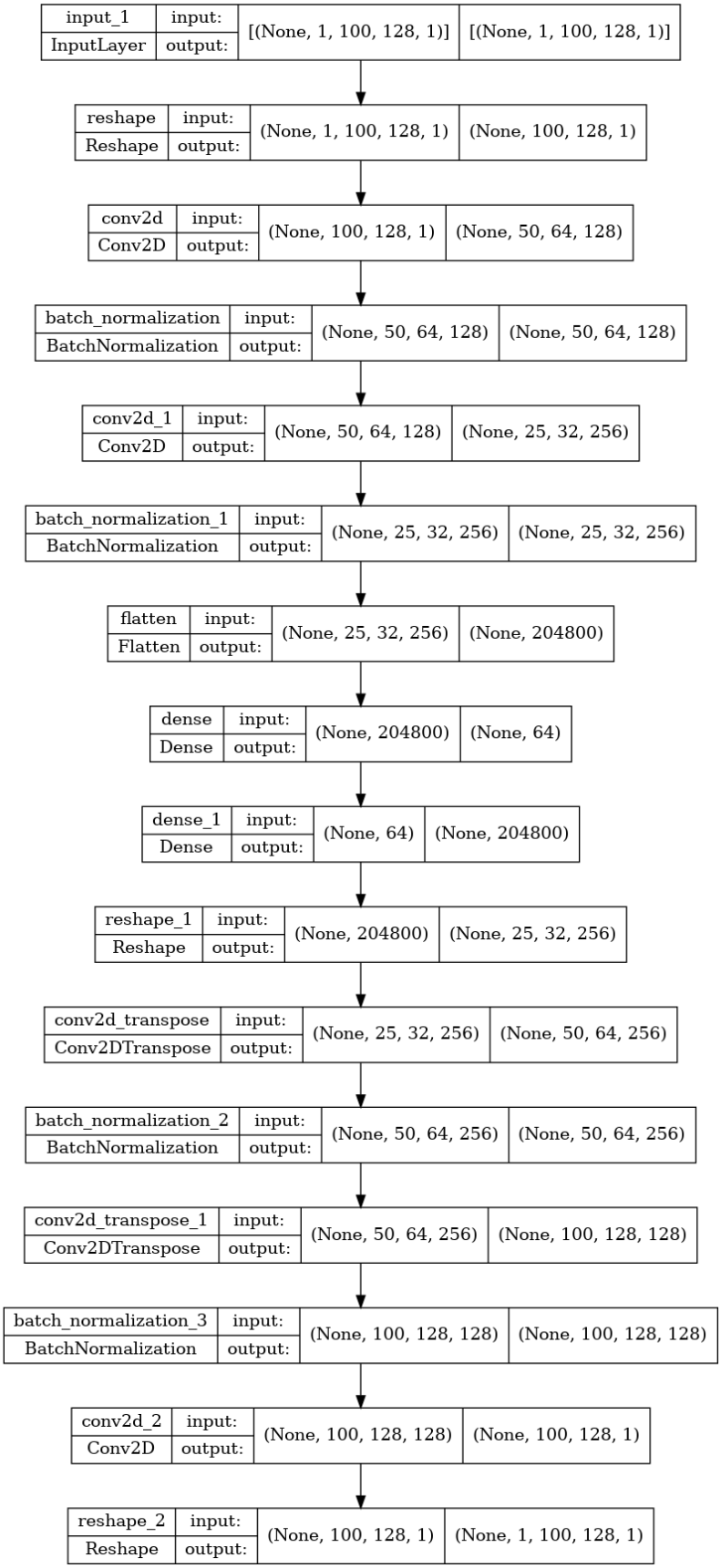


Figure 8. Model architecture of the Convolutional Autoencoder.

convolutional layer to obtain relevant features of the image and is then fed to an LSTM cell. The input of this model is an image and the output is numeric.

The model architecture is composed of three 2D convolutional layers with a kernel size of 3×3, stride 1, and padding 1. Every layer is followed by a ReLU activation function, a max pooling layer, and a dropout layer. The output of the convolutional layers is fed to a flatten layer to make it ready for the LSTM cell. The detailed architecture developed in TensorFlow is presented in Fig. 9.

Video-frame prediction (ConvLSTM)

The video-frame prediction model is composed of ConvLSTM layers [24]. Every layer is followed by a ReLU activation function, a batch normalization layer, and a dropout layer. First, the image is down-sampled similarly to an encoder network, and later the image is up-sampled to its original dimension, similarly to a decoder network. The input is an image, and the output is also an image. The detailed architecture developed in TensorFlow is presented in Fig. 10.

Long Short-Term Memory (LSTM)

The Long Short-Term Memory (LSTM) is utilized as a numeric-to-numeric approach. The optimized version of the model for this dataset consists of an input length of 50, 20 neurons and 1 LSTM layer. The detailed architecture developed in TensorFlow is presented in Fig. 11.

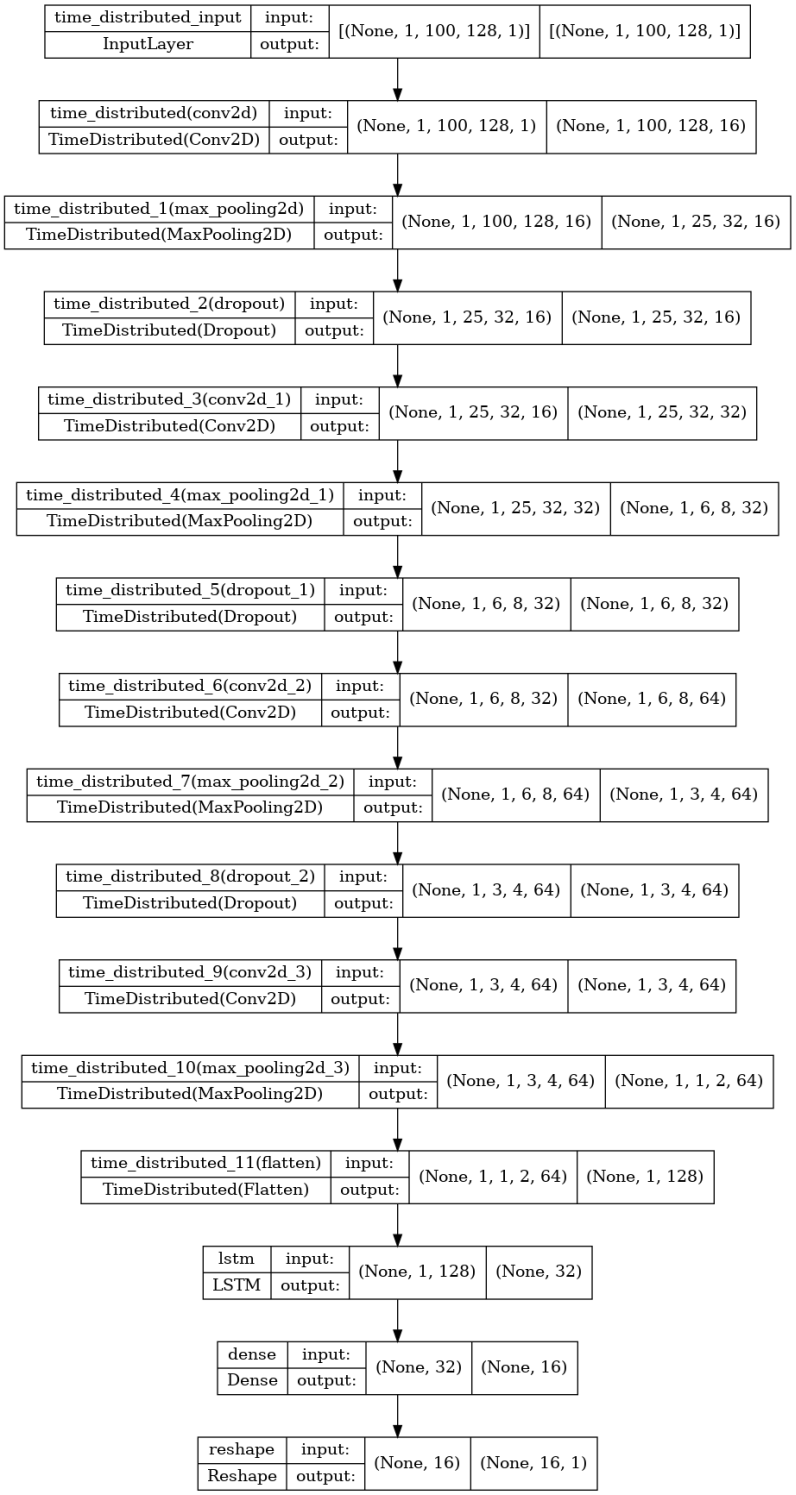


Figure 9. Model architecture of the Long Recurrent Convolutional Network (LRCN).

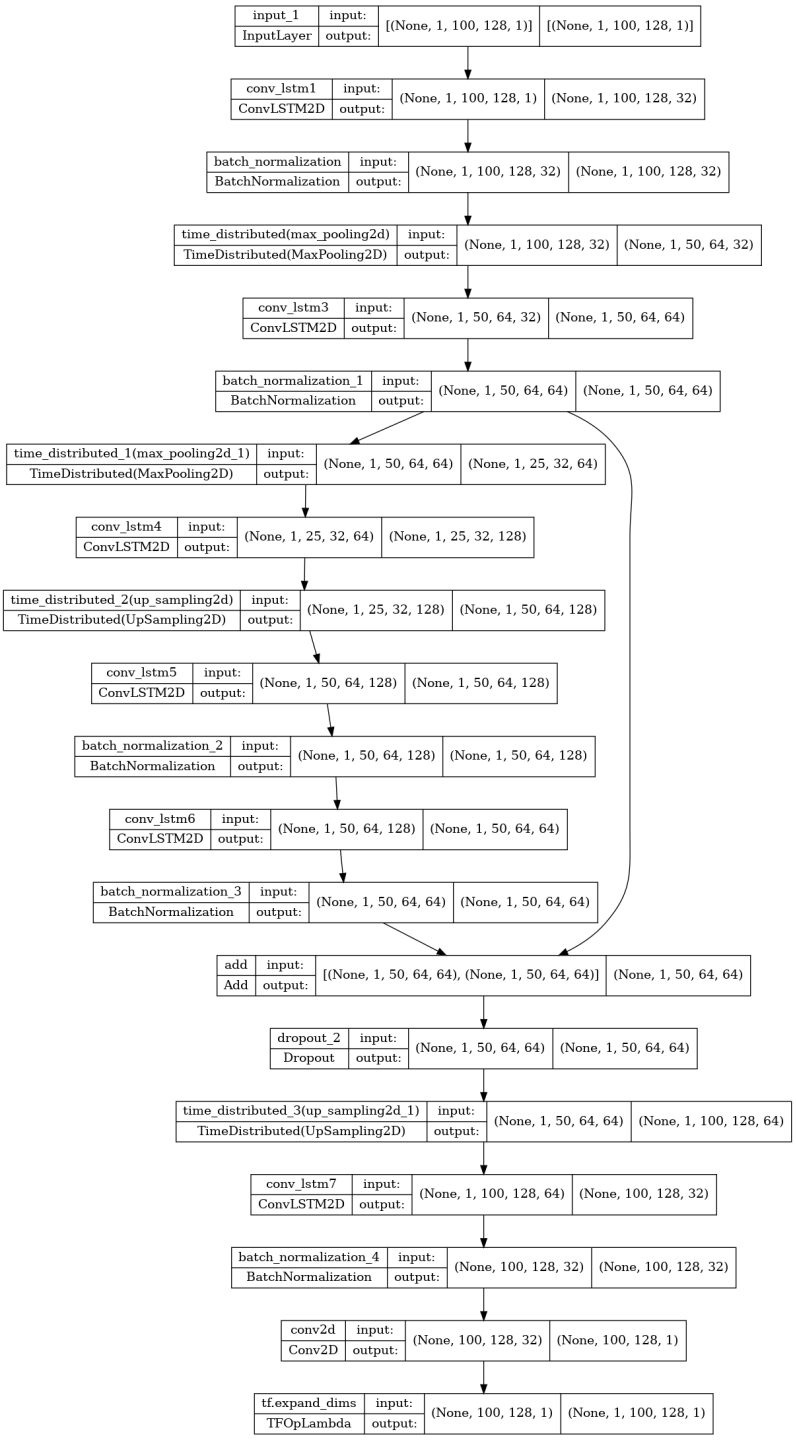


Figure 10. Model architecture of the video-frame model (ConvLSTM).

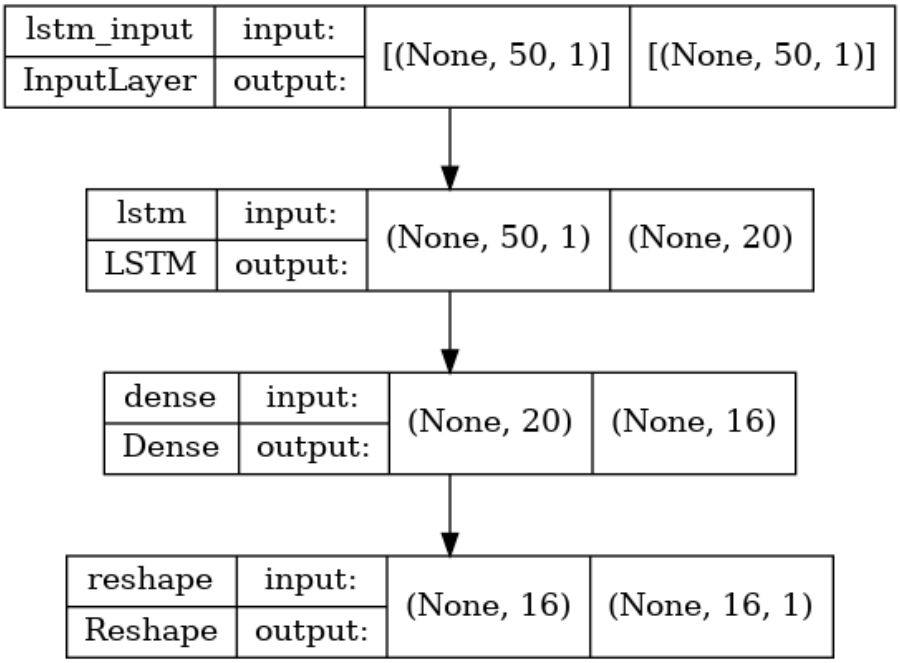


Figure 11. Model architecture of the LSTM model.

5. Data Acquisition and Exploration

In this chapter, the dataset is presented, explored, and depicted. The preprocessing necessary to deal with the data is also presented. Finally, a clustering algorithm is proposed to make relevant groups of the data.

One of the goals of this thesis is to validate that image-driven methods could be used to forecast future cloud resources utilization. In this context, it is key to find a dataset from real traces of a cloud datacenter. Previous work in time series forecasting [16, 17] uses synthetic datasets for time series forecasting. However, a certain model might perform appropriately in synthetic datasets and might not be able to capture the intrinsics of a real dataset. Thus, it is key to find traces of a cloud environment. The dataset GWA-T-12 Bitbrains [40] presented in this chapter has been selected for this work, as it has been widely used [41, 42, 43, 44, 45, 46, 47]. Other datasets such as Google Cluster Workload Traces [48] and Alibaba Cluster Trace Program [49] have been also used in the field.

5.1 Dataset Description

The dataset GWA-T-12 Bitbrains [40] contains the performance metrics of 1,750 Virtual Machine from a distributed datacenter from Bitbrains, which is a service provider that specializes in managed hosting and business computation for enterprises. Customers include many major banks (ING), credit card operators (ICS), insurers (Aegon), etc.

Each file contains the performance metrics of a VM. These files are organized according to traces: fastStorage and Rnd.

The first trace, *fastStorage*, consists of 1,250 VMs connected to SAN storage devices. The second trace, *Rnd*, has 500 VMs attached to either fast SAN devices or significantly slower Network Attached Storage (NAS) devices. Due to the better performance of the storage associated with the *fastStorage* machines, the *fastStorage* trace contains a higher percentage of application servers and compute nodes than the *Rnd* trace. In contrast, we see a higher proportion of management machines in the *Rnd* trace, which merely require storage with lower performance and less frequent access. [50]

The format of each file is row-based, where each row represents an observation of the performance metrics. The variables of the dataset are:

- Timestamp: number of seconds since 1970-01-01.
- CPU cores: number of virtual CPU cores provisioned.
- CPU capacity provisioned (CPU requested): the capacity of the CPUs in terms of MHZ, equals the number of cores x speed per core.
- CPU usage: in terms of MHZ.
- CPU usage: in terms of percentage
- Memory provisioned (memory requested): the capacity of the memory of the VM in terms of KB.
- Memory usage: the memory that is actively used in terms of KB.
- Disk read throughput: in terms of KB/s
- Disk write throughput: in terms of KB/s
- Network received throughput: in terms of KB/s
- Network transmitted throughput: in terms of KB/s

For the sake of visualization, an individual plot of every feature of the dataset for a single VM is presented in Fig. 12.

The monitoring and management tools provided by VMware, such as the vCloud suite, are utilized to collect traces. The vCloud Operation tool captures 10 performance metrics per VM for each trace, which are sampled every 5 minutes. The information was gathered during August and September of 2013. The traces, when combined, collect data for 1,750 nodes with over 5,000 cores and 20 TB of memory, and operationally amass over 5 million CPU hours over four months; consequently, they are long-term and large-scale time series.[50]

This thesis is focused on the analysis of the *fastStorage* trace, henceforth the results presented are about this trace.

From this dataset, the CPU utilization has been selected as the feature to forecast as it is the bottleneck of the Datacenter. Other features of the cloud datacenter, such as memory usage or disk throughput, could be model with simpler models and are out of the scope of this thesis. Future work will explore the influence of these features. In this chapter, the dataset is explored and characterized. Furthermore, a clustering approach is presented to find similar VMs within the same datacenter.

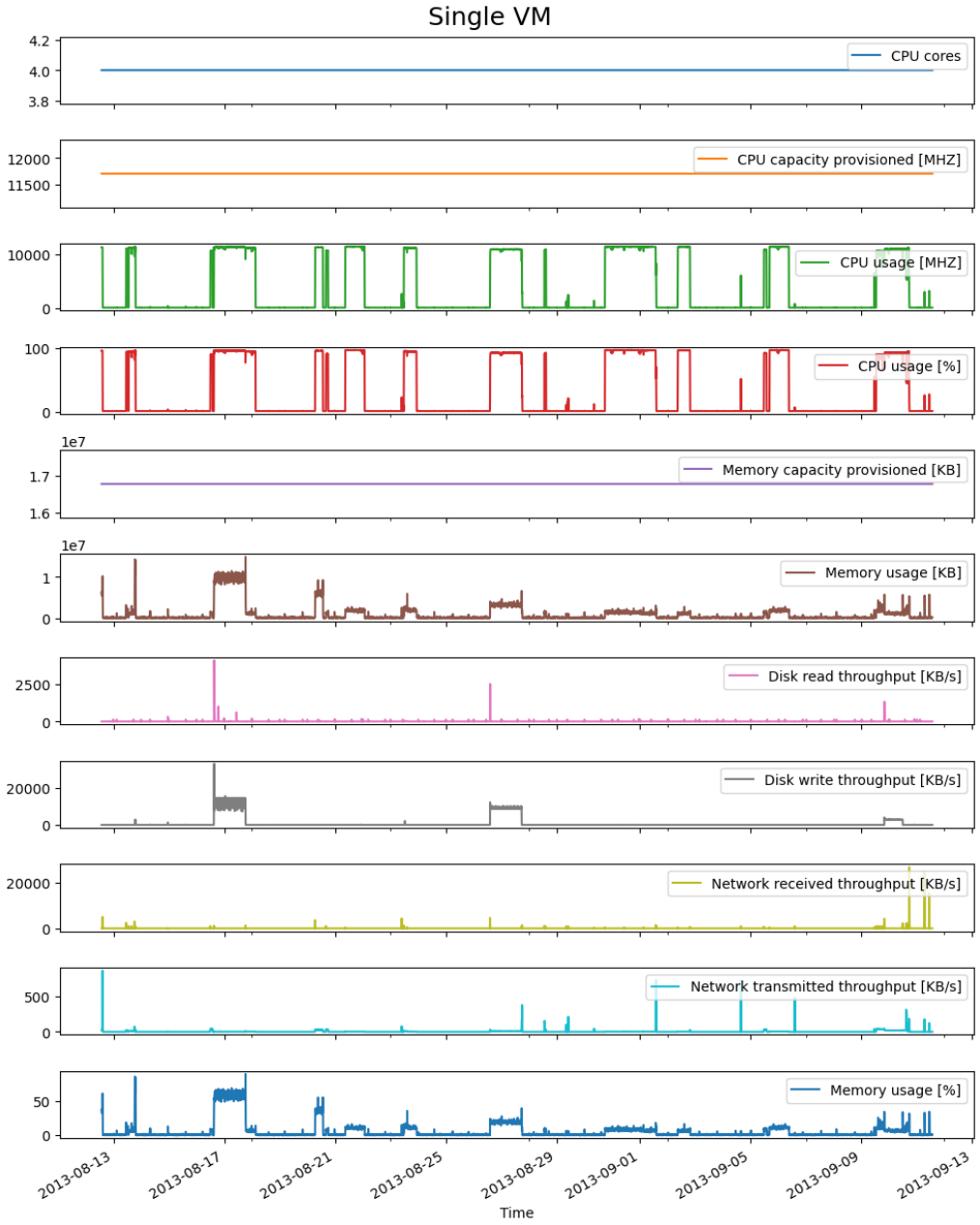


Figure 12. Individual plot of every feature for one selected Virtual Machine (VM) of the Bitbrains dataset.

5.2 Data Characterization

The first step is to correctly load the data in a proper format. Each VM is a multivariate time series with 10 features and the Timestamps. Each of them is saved in a *.csv* file.

To correctly process the data, parsing the dates is required. The official website of the dataset [40] states that the timestamps are the number of milliseconds since 1970-01-01, however, this data is not consistent with the official paper [50] presented on the same website. Furthermore, whether the unit of the timestamp is milliseconds, the duration of

the time series would be 43 minutes and not 1 month as stated in the description of the dataset. Hence, we assume that there is a typo in the website and the unit of the timestamp is seconds. Now, the first date of the dataset is in August of 2013 as is also stated in the description of the dataset [40]. Thus, the variable timestamp is parsed as seconds since 1970-01-01 and set as the index of the time series.

One additional variable *memoryUtilization* in percentage (%) is computed for each time series to have equivalence with the CPU utilization in percentage (%). This new feature is computed as follows:

$$memoryUtilization = 100 \cdot \frac{Memory\,usage}{Memory\,provisioned} \tag{5.1}$$

It is noteworthy that some timestamps (indexes) are repeated in some of the VM. In this case, to avoid duplicate entries, the duplicate timestamps are grouped in a single one performing the arithmetic mean.

When a time series is presented in this thesis, usually together with the raw data (in blue), the series after performing some filters is also presented to observe the trend of them and have a better understanding of the whole picture. Specifically, an Exponential Moving Average (EMA) filter is utilized.

The EMA is a first-order infinite impulse response filter that applies weighting factors that decrease exponentially. The weighting for each older data decreases exponentially, never reaching zero.

The EMA for a series Y may be calculated recursively [51]:

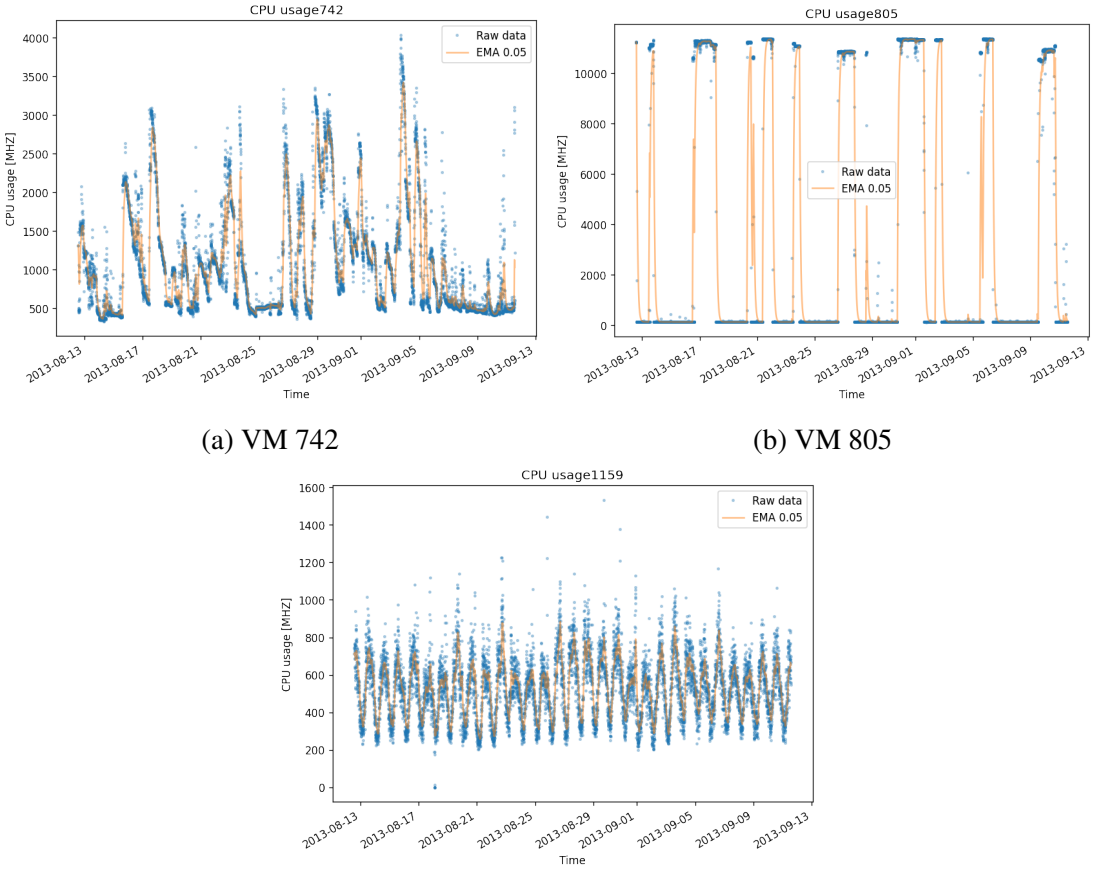
$$S_t = \begin{cases} Y_0, & t = 0 \\ \alpha Y_t + (1-\alpha) \cdot S_{t-1}, & t > 0 \end{cases} \tag{5.2}$$

Where:

- The coefficient α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster.
- Y_t is the value at a time period t.
- S_t is the value of the EMA at any time period t.

The optimal hyper-parameters of the filter have been empirically determined. In particular, an alpha (α) value of 0.05 is determined for the EMA filter.

There are 1250 VMs, thus, the individual analysis of each one individually is very time-consuming and does not offer much information. Furthermore, the behavior of each VM may vary significantly, as could be observed in Fig 13.



(a) VM 742

(b) VM 805

(c) VM 1159

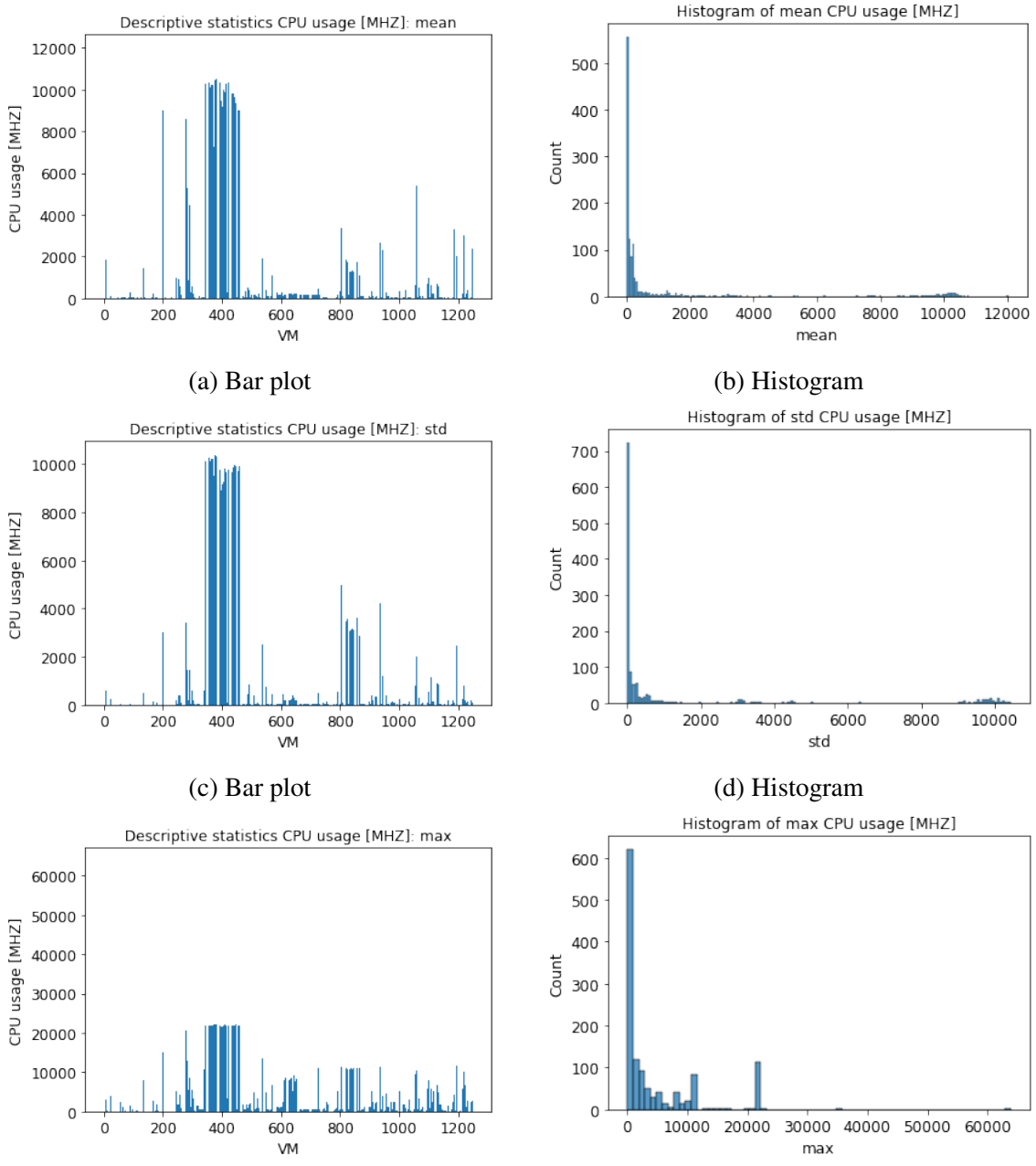
Figure 13. CPU utilization [MHz] across Virtual Machines exhibiting different trends.

Due to the diversity of VMs within the same data center, an statistical analysis of the individual VMs is needed. Thus, some descriptive statistics are computed for each VM and summarized in the next figures to have a better understating of the dataset.

Specifically, the statistics computed for each VM are:

- mean
- standard deviation (std)
- maximal value (max)
- minimum value (min)
- cumulative sum (sum)

The statistics are computed for the key feature of the dataset, the CPU usage [MHz]. For each statistic, a bar plot where the x-axis corresponds to the number of the VM and the y-axis is the CPU usage [MHz] and a histogram are presented (see Fig. 14 and Fig. 15).



(a) Bar plot

(b) Histogram

(c) Bar plot

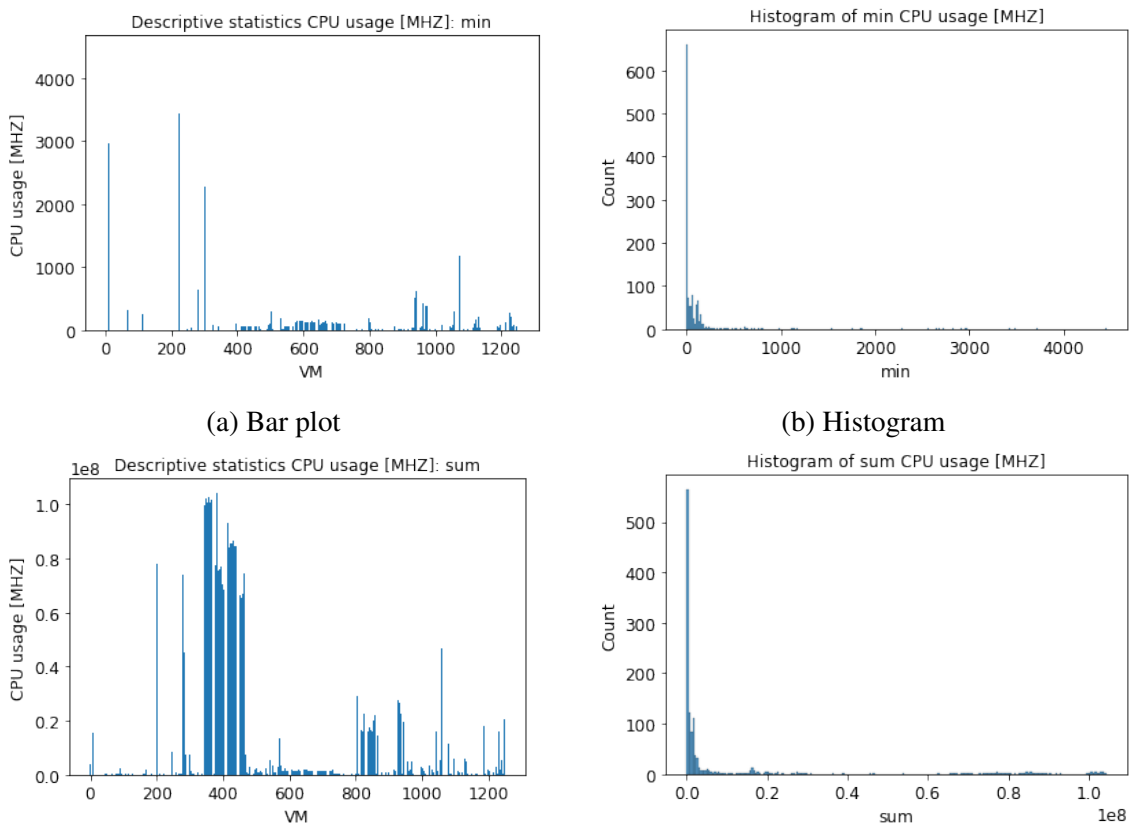
(d) Histogram

(e) Bar plot

(f) Histogram

Figure 14. Descriptive statistics (mean, standard deviation and max value) of the distribution of the CPU utilization across the Virtual Machines of the Datacenter.

Observing the histograms presented in the aforementioned figures, one should note that there are a large number of VMs (around 500-600) whose statistics are always very close to 0. Thus, they are acting as noise and are not relevant for the data center as a whole. One way to "clean" the dataset is to find thresholds for the mean, std, etc., to "filter" the least important VMs and train the algorithm only on the relevant ones. The VMs considered as



(a) Bar plot

(b) Histogram

(c) Bar plot

(d) Histogram

Figure 15. Descriptive statistics (min value and sum) of the distribution of the CPU utilization across the Virtual Machines of the Datacenter.

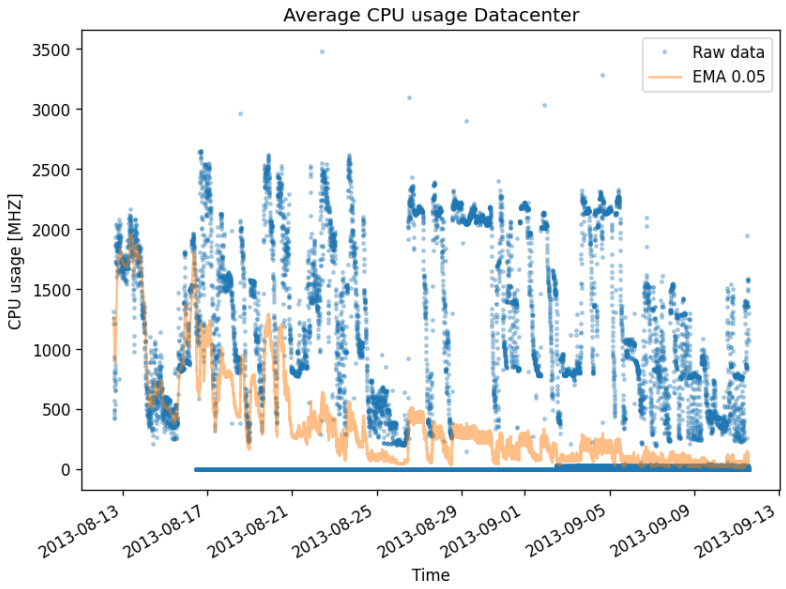
important are the ones that have relevance in the overall performance of the data center (i.e., has a relevant mean CPU utilization, has a relevant variation of CPU utilization, or the max CPU utilization is relevant).

To have an overview of the data center as a whole (i.e., the addition of all the VMs), a new DataFrame of the whole data center is created. When concatenating a new VM to the data-center DataFrame, whether a timestamp is novel, it is appended to the DataFrame. On the flip side, whether a timestamp already exists, the new value of the variable is the mean of all the VMs present at this specific timestamp.

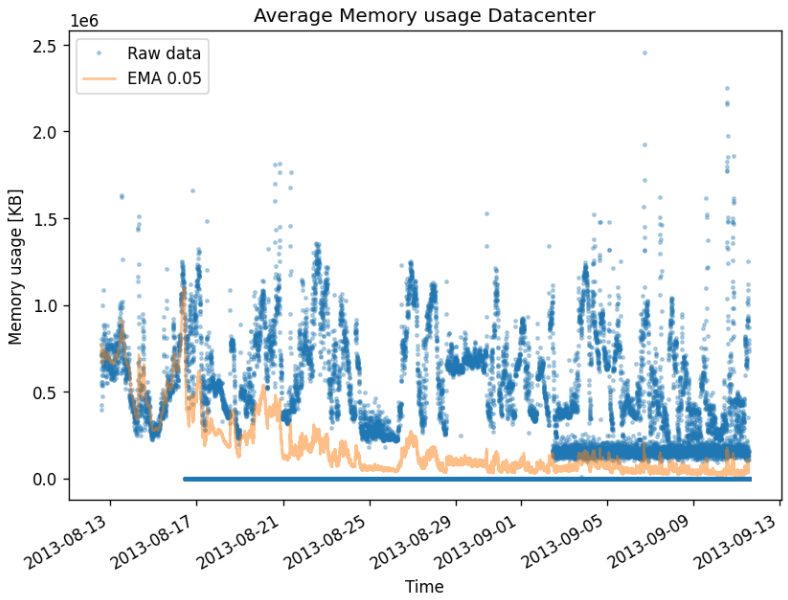
The time series of the data center of the CPU utilization [MHz] and memory utilization [KB] are presented in Fig. 16.

5.3 Data Clustering

Analyzing each VM individually is very time consuming but having only one time series of the whole data-center may dismiss too much information. Thus, finding clusters of VMs



(a) Average CPU utilization



(b) Average Memory utilization

Figure 16. Average CPU and Memory utilization of the whole datacenter of the Bitbrains dataset.

within the same data center is a valuable approach to leverage the potential of the dataset. Hence, a group or cluster of VMs could be handled as a whole towards finding an effective forecasting algorithm for the involved VMs within the cluster.

5.3.1 Data Preparation

Each VM contains 11 features and around 8000 data points and the data center has 1250 VMs. The feature space is too large to handle in a reasonable time, even using professional

hardware. Hence, dimensionality reduction algorithms are needed. There are two possible approaches: reducing the length of the time series and/or performing feature selection. Due to the high dimensionality of the dataset, both approaches are implemented.

To reduce the length of the time series, the series is resampled to a specific length using 1-D interpolation. The operation is performed on the function `TimeSeriesResampler` from the package `tslearn`. The optimal length of the time series is determined through trial and error and is found to be 500 timestamps. The results of the shortened time series of a VM and the comparison with the original time series are presented in Fig. 17. One could observe how the shape of the time series is maintained after the resample operation. Although some information is lost, the shortened time series is valuable enough for the training stage.

After performing this operation, the dimensional of the dataset is reduced by $x16$ factor.

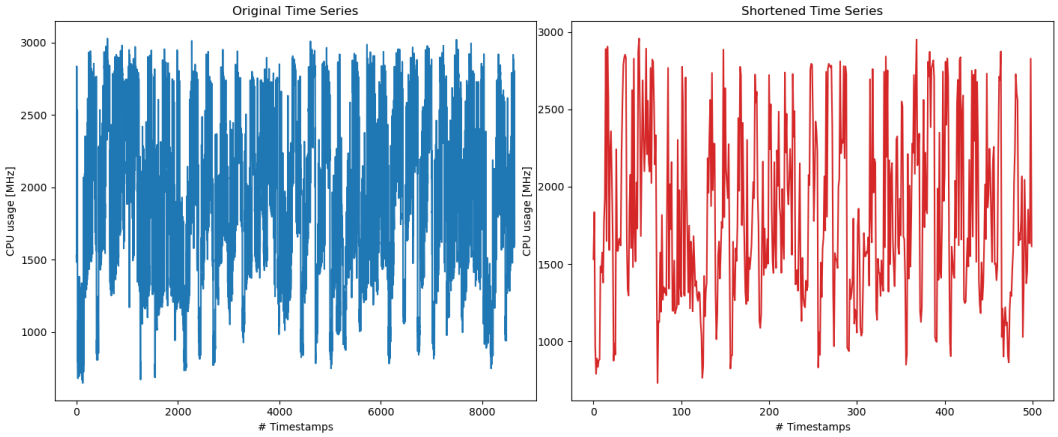
The other approach is to perform feature selection. After visually exploring the different features of the dataset, it is impaired that the most informative ones are *CPU usage [MHZ]* and *Memory usage [KB]*. Additionally, these features are the bottleneck of the data center and the ones with more interest in this thesis. To explore what are the best features for clustering, the scenarios analyzed are:

- Feature(s): CPU usage [MHZ]
- Feature(s): Memory usage [KB]
- Feature(s): CPU usage [MHZ] and Memory usage [KB]

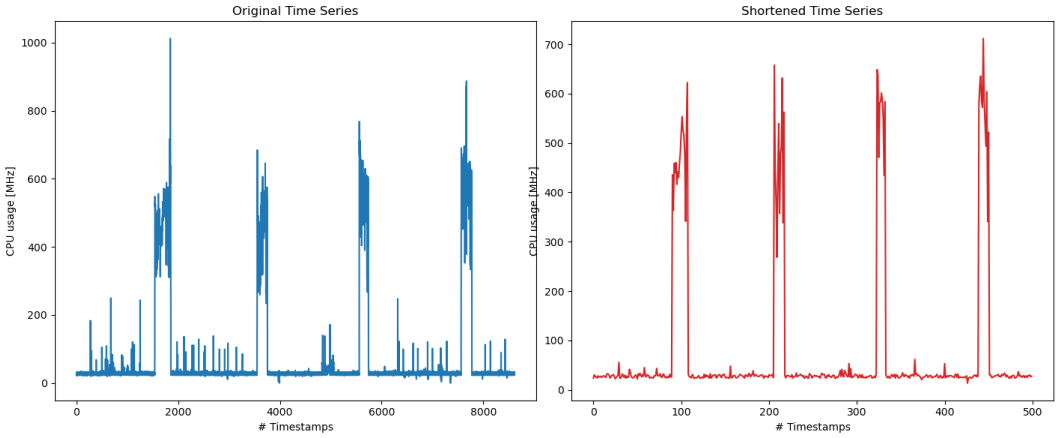
Data cleaning

As it has previously stated in Section 5.2, the histograms presented in Fig. 14, and Fig. 15 show that there are a large number of VMs (around 500-600) whose statistics are always very close to 0. Thus, they are acting as noise and are not relevant to the data center as a whole. To "clean" the dataset, it is necessary to find thresholds for the mean, std, etc., to "filter" the least important VMs and train the algorithm only on the relevant ones. One VM is considered relevant whether it is important in the operation of the data center. It is important if the average is higher than a certain threshold, if the variation over time is large or if the VM has some peak of utilization (i.e., large max value).

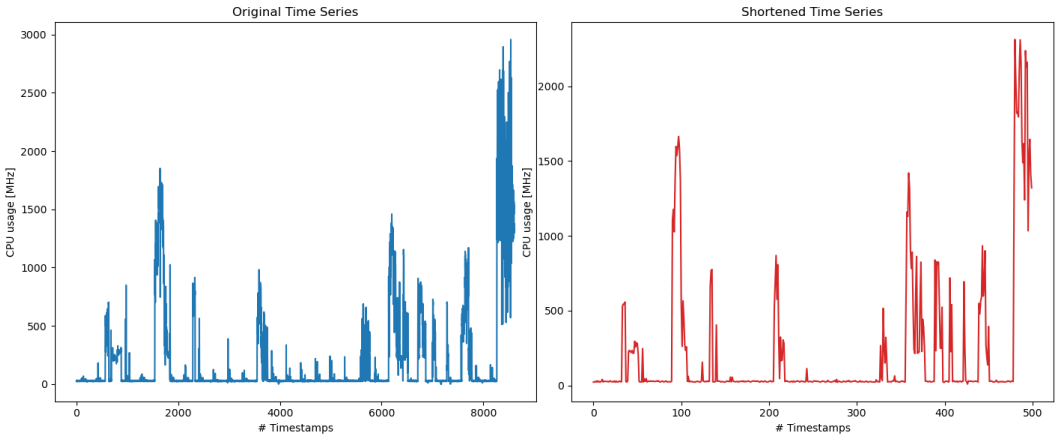
It has been empirically determined and visually validated that the following thresholds properly split the dataset into "relevant" and "non-relevant" VMs:



(a)



(b)

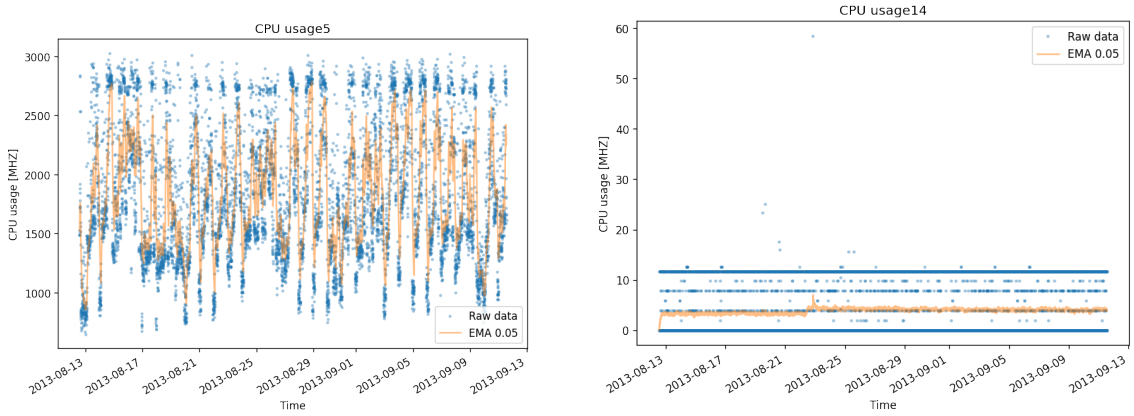


(c)

Figure 17. CPU utilization of various Virtual Machines. Original data compared to shortened time series (Clustering prepossessing).

- Mean threshold: 100 MHz
- Std threshold: 50 MHz
- Max threshold: 1000 MHz

The above-mentioned thresholds are *OR* conditioned (i.e., whether one or more of the conditions is satisfied, the statement is satisfied), thus all the relevant VMs are considered. An example of a "relevant" VM and "non-relevant" VM is presented in Fig. 18. Looking closely at the y-axis, it is obvious that one VM is working in a relevant range (1000-3000 MHz) while the other is insignificant (0-10 MHz). After applying the filter, the training set is composed of 768 VMs. Thus, there are 482 "non-relevant" VMs.



(a) Relevant VM

(b) Non-relevant VM

Figure 18. Example of CPU usage [MHz] for two Virtual Machines that exhibit very different behavior.

5.3.2 Clustering with K-Means

Clustering is an unsupervised machine learning where an algorithm groups data points in such a way that the ones in the same group (or cluster) are more similar to each other than to those in other groups (or clusters). The nature of the time series makes clustering a challenging task because each data point is an ordered sequence. Furthermore, the varying length of the series in this dataset causes some traditional metrics used for clustering (e.g., Euclidean distance) become unsuitable. Thus, some considerations should be taken into account.

K-means is a very common clustering algorithm that creates clusters of data by splitting samples into k groups and minimizes within-cluster variances (squared Euclidean distances) [52]. However, the standard k-means algorithm cannot be applied directly. First, it is needed to make some modifications to the algorithm. Instead of the Euclidean distance, the distance between the series is computed with DTW.

Selection of the Number of Clusters

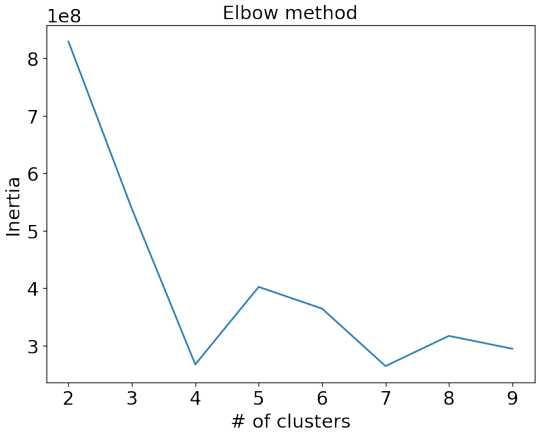
Representative based clustering techniques such as k-means need the user to specify the

number of clusters k to be generated. To determine the optimal number, the elbow method and the silhouette score are the most common approaches.

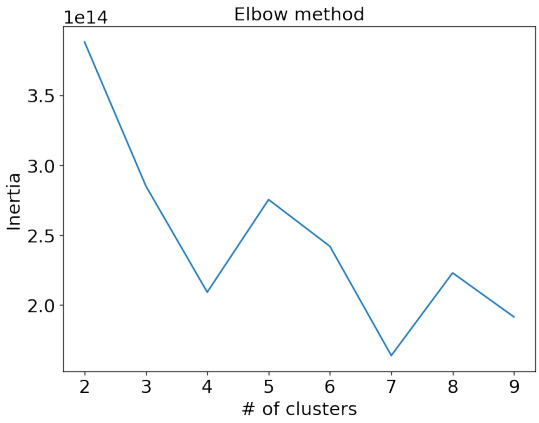
Elbow Method

The elbow method is a technique used for determining the number of clusters within a dataset. The method consists of plotting the inertia (sum of squared distances of samples to their closest cluster center) as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. [53]

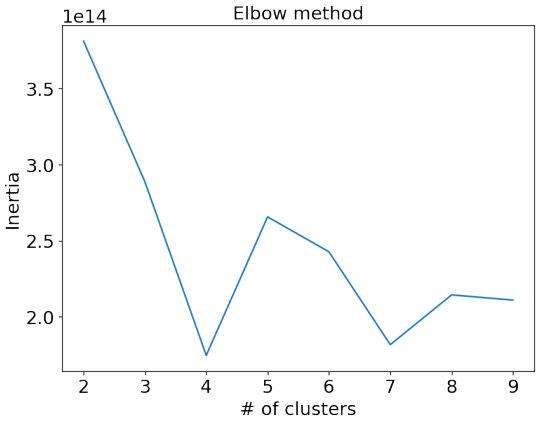
As mentioned in Section 5.3.1, feature selection is performed to reduce the dimensionality of the dataset. Thus, the elbow method is computed when the only feature is CPU usage [MHZ] (see Fig. 19b), when the feature is memory usage [KB] (see Fig. 19b) and when both features are considered (see Fig. 19c).



(a) Feature: CPU usage



(b) Feature: Memory usage



(c) Features: CPU usage & Memory usage

Figure 19. Elbow method plots for CPU usage, Memory usage and CPU usage & Memory usage as feature(s).

From the figures, one could be observed that there are slight differences between the plots

for the different figures. However, there is not a clear elbow in the figures. Fig. 19a shows the clearest "elbow shape" and the optimal number of clusters (k) is 4. The clustering output for $k = 4$ is presented in Fig. 20. For the sake of simplicity, only the CPU usage [MHz] is presented because it is the key feature of the dataset. The left figures present a scatter plot of all the VMs presented in the cluster. The VMs are overlapped one over another. The right figures, present a scatter plot of the average over all VMs within the specific cluster and a filtered time series to observe the trend of the cluster. It is noteworthy that there is a very big cluster (573 VMs) without a clear shape and is very "messy" and 3 other clusters with a lot of similarity within the cluster and clear shape.

Silhouette Coefficient

The silhouette coefficient or silhouette score is a metric used to validate the goodness of a clustering technique. The overall silhouette score may be also used to determine the optimal number of clusters. The silhouette coefficient is computed as follows:

$$s(i) = \frac{D^{out}_{min_i} - D^{in}_{avg_i}}{\max\{D^{out}_{min_i}, D^{in}_{avg_i}\}} \tag{5.3}$$

where $D^{in}_{avg_i}$ is the average distance of the point x_i to the points within the cluster it belongs to. The average distance of point x_i to the points of each cluster is also computed. Let $D^{out}_{min_i}$ be the minimum of these average distances. The overall silhouette score is the average of the individual points coefficients $s(i) \in (-1, 1)$. Large positive numbers indicate highly separated clusters. Values close to 0 indicate that the means clusters are indifferent, or the distance between clusters is not significant. Large negative values indicate that clusters are assigned in the wrong way. One advantage of this coefficient is that the absolute values provide a good intuitive feel of the quality of the clustering. [35]

Analogous to the previous section, with the elbow method, the silhouette score for different numbers of clusters is computed for different features. The results are presented in Fig. 21. The three plots present that 2 is the optimal cluster. Furthermore, the silhouette score for 2 clusters is around 0.83, thus, the goodness of clustering is high.

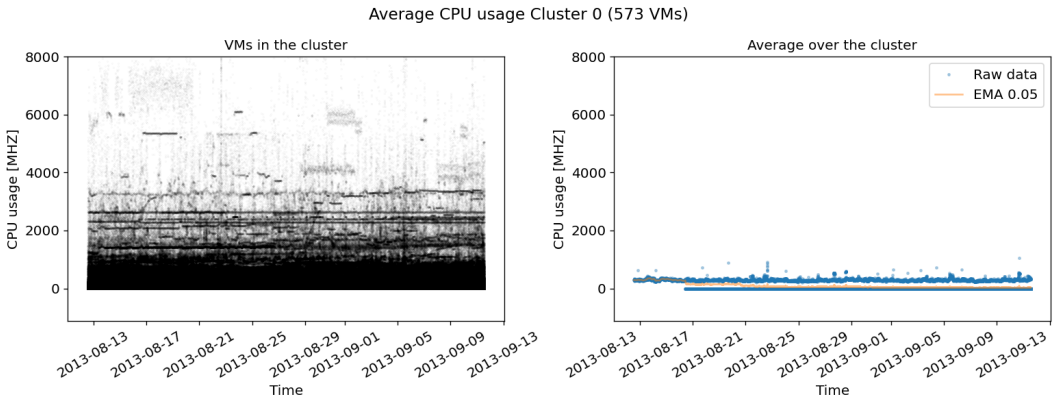
The clustering output for $k = 2$ is presented in Fig. 22. The left figures present a scatter plot of all the VMs presented in the cluster. The VMs are overlapped one over another. The right figures, present a scatter plot of the average over all VMs within the specific cluster and a filtered time series to observe the trend of the cluster. Again, the same phenomenon is present, there is one very big cluster (656 VMs) without a clear shape and a smaller

cluster where the different time series are very similar and a clear trend is shown.

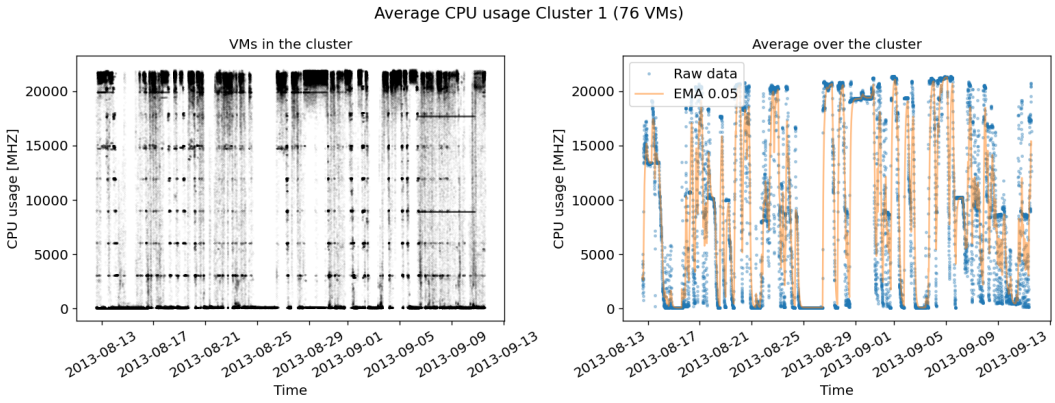
5.3.3 Clustering Effectiveness

The goal of clustering in this thesis is to find groups of VMs that have a similar pattern over time. Thus, these VMs could be grouped and treated as such to develop a forecasting model. Whether the pattern of VM differs significantly from one to another, the model would not be capable of effectively forecasting future timestamps. Considering the main goal of clustering, the goal has not been achieved. As shown in Fig. 20a and Fig. 22a, there is one big cluster with 573 and 656 VMs respectively. This implies that more than 70% of the VMs are in one cluster. This would not be a problem whether the shape of all VMs in this cluster would be similar. However, this is not the case, and it is not possible to extract a clear trend from this cluster. Furthermore, to avoid that "non-relevant" VMs hindering the clustering, a preprocessing filtering technique is performed, albeit the problem is not prevented. The increase in the number of clusters has also been explored to prevent the phenomenon without success. Even increasing the number of clusters up to 10, there is always one cluster that contains at least 70% of the VMs without being too noisy to have a clear shape or trend.

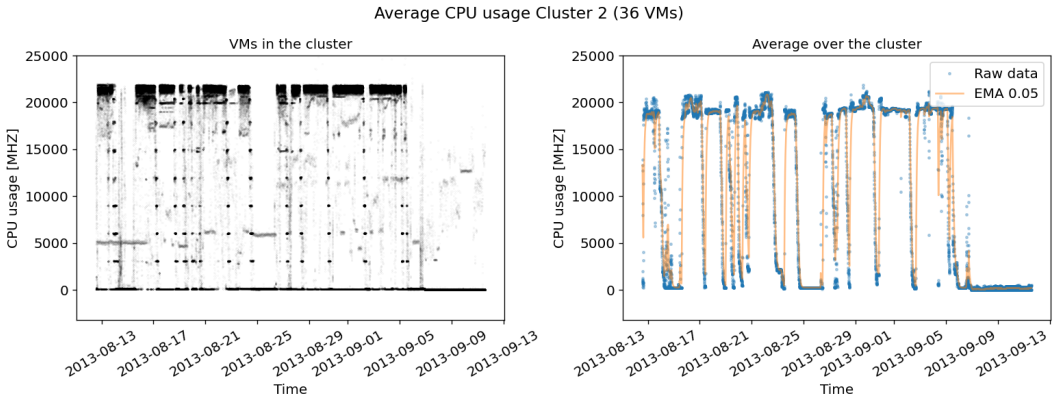
Nevertheless, for the sake of the experiments, a cluster of VMs with similar patterns should be created. K-means has not produced the expected outcome, so we have manually created a cluster ofVMs that follow a defined trend similar to a sinusoid. The periodicity of this series is around 288 timestamps or 1 day. The resource utilization is higher during the day and reduced during night hours. The VMs presented in this manually created cluster are shown in Fig. 23, Fig. 24, Fig. 25 and Fig. 26.



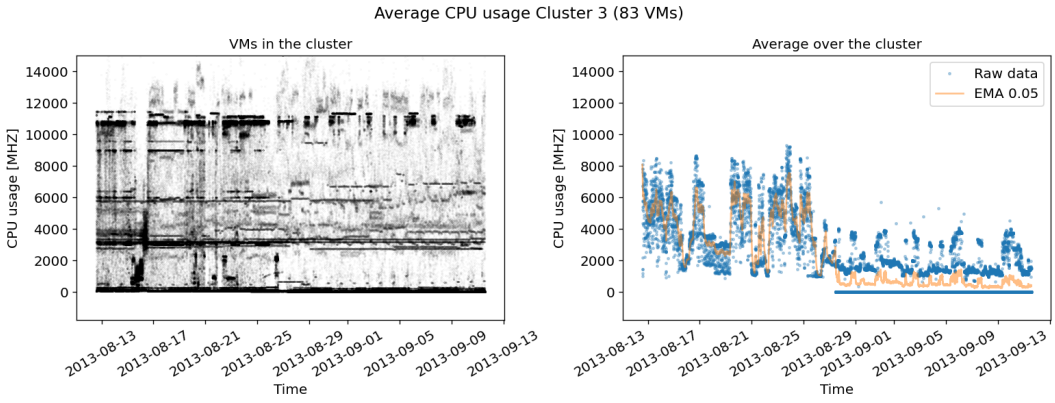
(a) Cluster 1



(b) Cluster 2

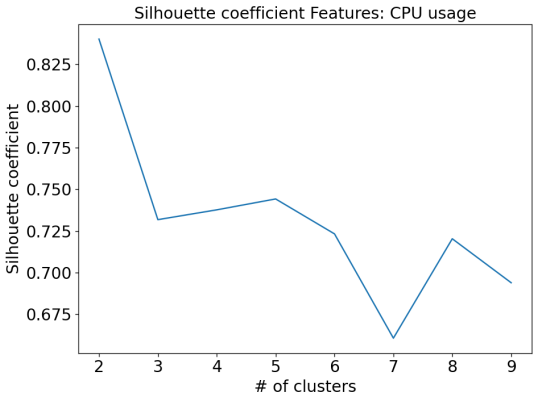


(c) Cluster 3

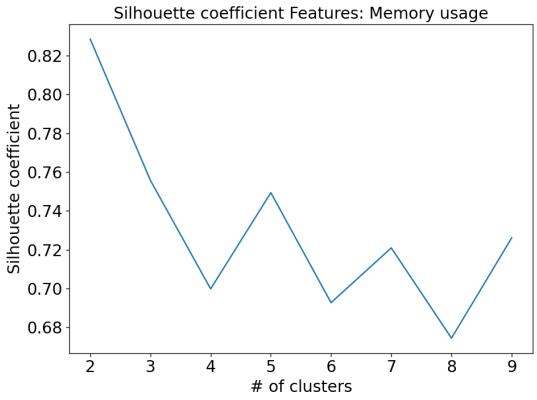


(d) Cluster 4

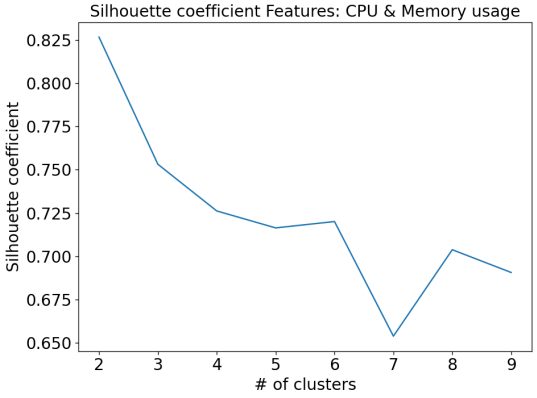
Figure 20. Output of K-means clustering $(k = 4)$. CPU usage [MHz]. Filtered Virtual Machines.



(a) Feature: CPU usage

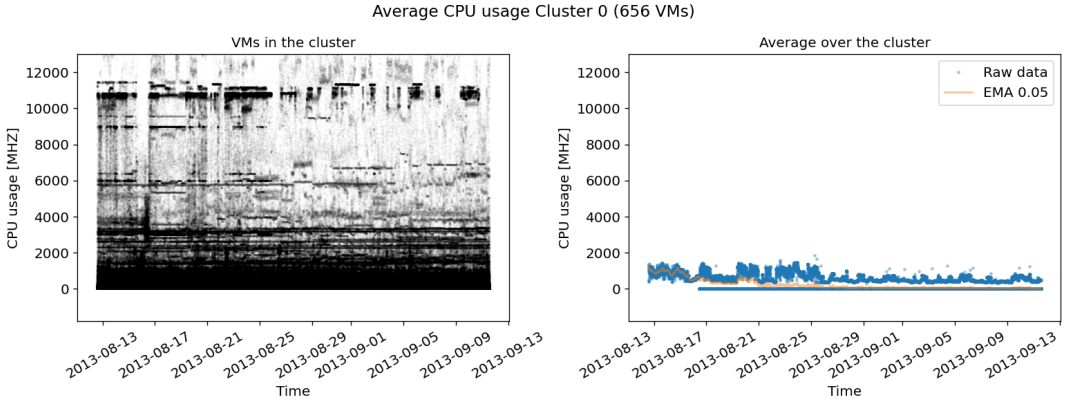


(b) Feature: Memory usage

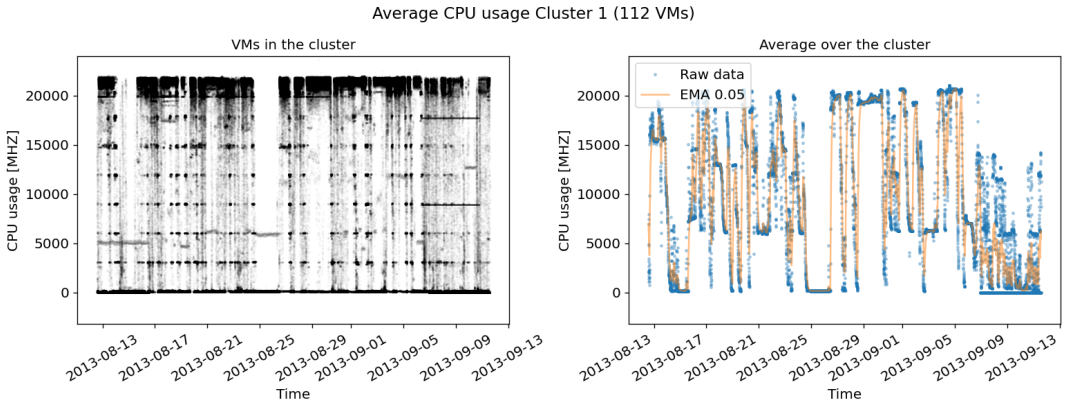


(c) Features: CPU usage & Memory usage

Figure 21. Silhouette score for CPU usage, Memory usage and CPU usage & Memory usage as feature(s).



(a) Cluster 1



(b) Cluster 2

Figure 22. Output of K-means clustering $(k = 2)$. CPU usage [MHz]. Filtered Virtual Machines.

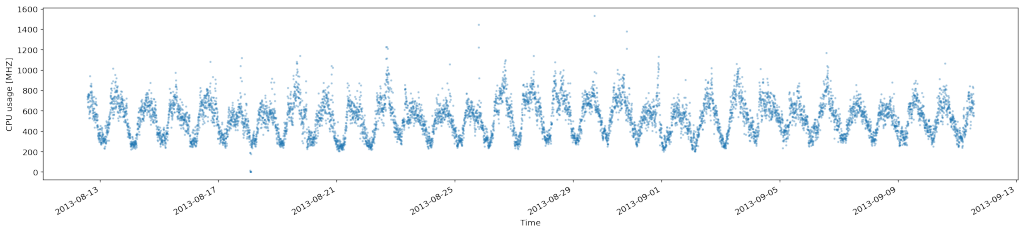


Figure 23. CPU utilization of Virtual Machine 917.

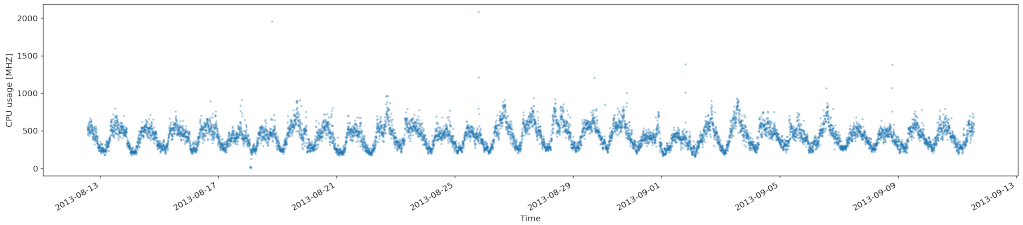


Figure 24. CPU utilization of Virtual Machine 340.

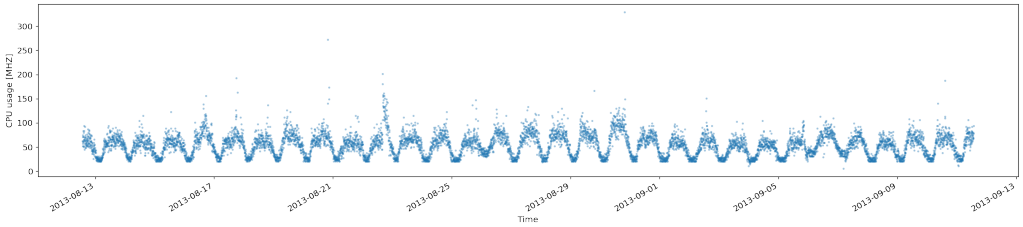


Figure 25. CPU utilization of Virtual Machine 1081.

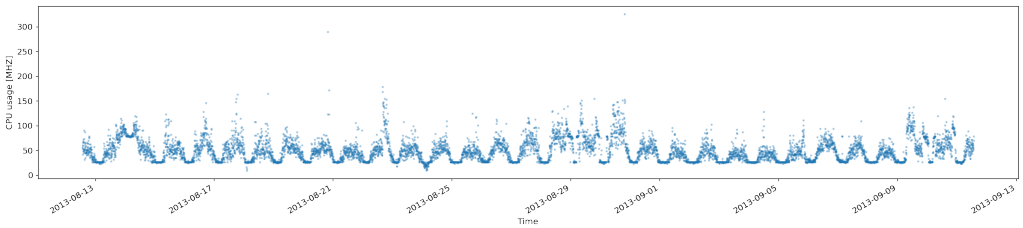


Figure 26. CPU utilization of Virtual Machine 555.

6. Data Processing

Processing data properly is one of the most important parts of a data science project. Having a good understating of the dataset to then process the data is key to the success of a machine learning model. Specifically, when dealing with data where time is involved, this process is even more important. In this chapter, we discuss how the data are processed for both the image-driven and the numeric approaches.

6.1 Image-based Machine Learning

Frequently, when engineers or data scientists are given tables or numerical data, they rely significantly more on visualizing the data through different types of graphs rather than looking at the numbers directly to gain a high-level comprehension of the data.

When laying out the underlying data in 2D graphics, visualizations convey spatial structure information [28] that is not present in the original data. Human eyes are skilled in capturing spatial structure or patterns in 2D images, which can aid in making better judgments or predictions. CNN [29] has been proven to have the ability to extract characteristics of local spatial regions, allowing computers to recognize spatial patterns such as those in object identification and recognition tasks, thanks to advances in deep learning and computer vision.

We propose to spatially layout numerical information in 2D graphics, inspired by how people benefit from 2D visualizations of numerical data. Then, for time series forecasting problems, we leverage CNNs, which were first investigated in non-image domains.

Given a 1D time series of a random variable $\{x_0, x_1, \ldots, x_t\}$ where $x_t \in \mathbb{R}$, the goal is to predict the values of the random variables in future timestamps $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$ where fh is the forecasting horizon. In this thesis, several timestamps are represented as an image (i.e., $\{x_0, x_1, \ldots, x_{input\_width}\} \mapsto I_t$ where $input\_width$ is the number of timestamps represented in one image). In this chapter, we discuss how to represent the series $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$ as a 2D image.

6.1.1 Image Creation

A good representation of the subset of time series $\{x_0, x_1, \ldots, x_{input\_width}\}$ as a 2D image is the key to a successful prediction. Whether the image does not represent the main pattern of the series, no algorithm will be capable of learning the intrinsics of the data and making good predictions.

Previous work in the field has explored the representation of a time series using recurrence plots [32] or GAF [31]. Nonetheless, these methods map a time series to an image ($\{x_0, x_1, \ldots, x_t\} \mapsto I_t$), but to this day it is not possible to map the image back to a time series ($I_t \mapsto \{x_0, x_1, \ldots, x_t\}$). During this thesis, an image-to-image approach is explored and thus the representation of time series as recurrent plots or GAF is not feasible.

Inspired by the work of Veloso et al. [16, 17], we are going to represent the time series as an image on a canvas where the x-axis is the time and the y-axis is the corresponding value of the series for this specific time.

To be able to map the image back to a time series again, it is necessary to create the 2D image in a specific manner. The height of the image (y-axis) has a fixed size of 100 and the image width is the number of timestamps represented in the 2D image.

To generate the image representation of the series, two methodologies have been explored, using either `numpy` or `matplotlib`. It is noteworthy that both approaches have a black background instead of a white one. This is common practice in the Machine Learning (ML) community, as could be observed in the handwritten digits dataset (MNIST) [54]. Additionally, the images have only one channel and they are binary images (i.e., each pixel is either black or white). Having grayscale images or more channels would only add unnecessary complexity to the model and thus would require more data and would have more chances to overfit.

`numpy` approach

To visualize the numeric data in an image using `numpy` or a purely mathematical approach, the following operations are performed:

$$y\_value = round(x_t \cdot 100) \tag{6.1}$$

where the input value at time step t, $x_t \in [0, 1]$. Thus, $y\_value \in [0, 100]$ corresponds to

the height of the y-value of this specific timestamp. This operation is repeated for every timestamp or x-value represented in the image. Fig. 34 illustrates the generation process. For the sake of simplicity, the height of the image is 10 instead of 100, but the process is analogous.

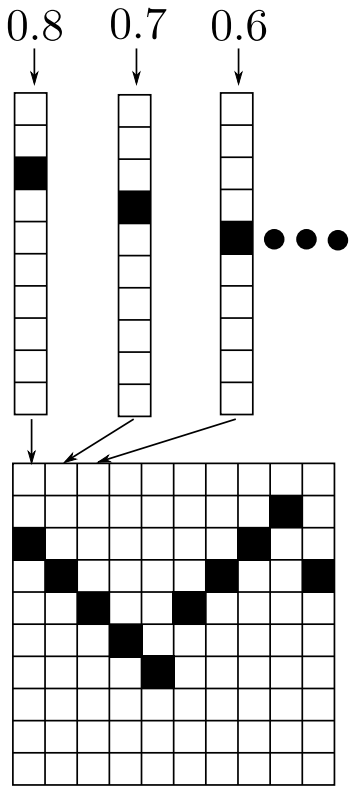


Figure 27. Illustration of the image generation process using `numpy`.

Afterward, a numpy matrix of zeros (black canvas) with shape (image_height, image_width) $\equiv$ (100, input_width) is created. Finally, in each column (one timestamp), the corresponding y_value determines the height of the pixel to be filled with 255 (white pixel). Thus, a 2D image representation of the image is generated. An example of the result is presented in Fig. 28.

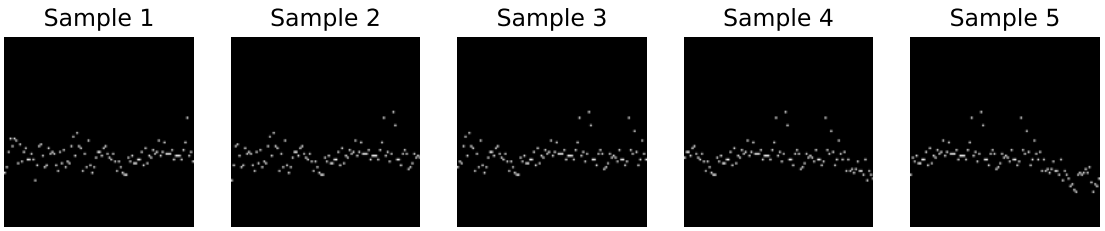


Figure 28. Example of an image generated using the numpy approach.

`matplotlib` approach

The other methodology consists of leveraging `matplotlib` to create the images. In this context, a black figure with a specific size (i.e., (image_height, image_width)$\equiv$(100,

input_width)) is created. Next, a subset of the time series $\{x_0, x_1, \ldots, x_{input\_width}\}$ is plotted in the figure. These data are saved in the buffer and decoded using `OpenCV2` to obtain a numpy matrix of the image. The last step is necessary to make the figure "understandable" for the ML algorithm. Note that there can be multiple bright (non-zero) pixels in each column due to anti-aliasing while plotting the images.

The subset of the series could be plotted as lines (observe Fig. 29) or as dots (see Fig. 30)

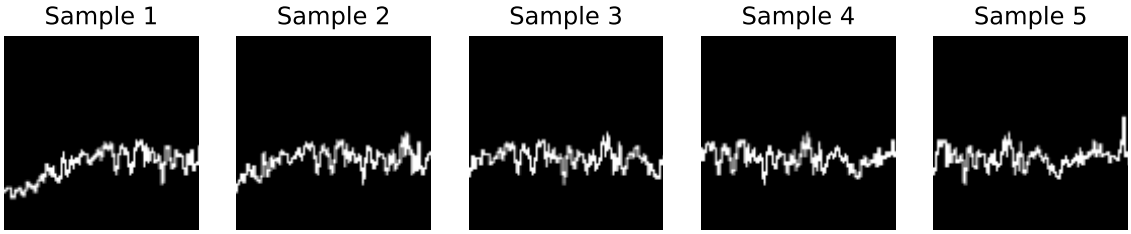


Figure 29. Example of an image generated using the matplotlib line approach.

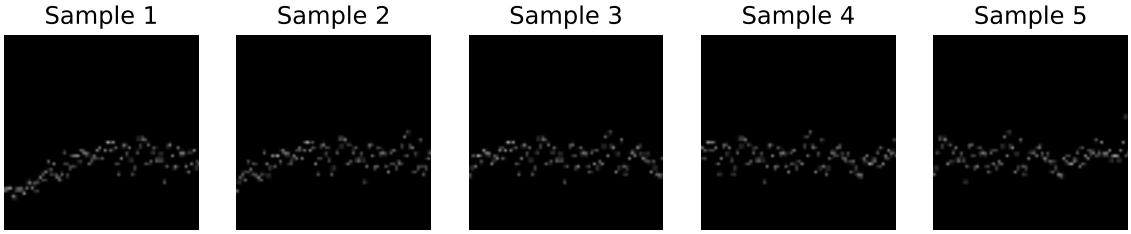


Figure 30. Example of image generated using the matplotlib dots approach.

Performance of the different techniques

The performance of the different techniques is analyzed in a vanilla convolutional autoencoder.

The baseline parameters for the convolutional autoencoder model are the following:

- dataset: VM 917 (see Fig. 23)
- forecasting horizon: 16
- image creation: matplotlib dots
- image size : 100x64
- input width: 64
- ratio: 1

A more mathematical way of using numpy is compared to the matplotlib approach (both dots and lines).

The different metrics for the 3 mentioned algorithms are presented in Table 1. According to the numeric metric (MAE, MAPE, RMSE, MASE) the matplotlib with dots slightly

Table 1. Prediction errors and performance metrics when changing the image creation method.

	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
Matplotlib dots	**79.64**	**15.07**	**102.357**	**1.345**	0.1	39608.301	102.168	1.368	198.057
Matplotlib lines	83.489	16.533	105.64	1.41	0.097	37302.619	105.358	1.39	198.057
Numpy	80.463	15.566	103.822	1.359	**0.114**	**36700.375**	**84.346**	**0.205**	198.057

outperforms the other approaches. Image-based metrics such as IoU or time series metrics such as DTW that better represent how the model fits the patterns of the data, show that the numpy approach is slightly better in this aspect. The training and inference times are also shorter in the numpy approach.

However, as is shown later, the matplotlib approach allows us to have multiple timestamps in a single image column, and thus is the selected option hereinafter.

Mapping Multiple Timestamps to a Single Pixel

As it has been previously mentioned, it is key that the image representation of the numeric values shows the main pattern on them. Whether this purpose is not achieved, the model would only see randomness and would not be able to make accurate predictions. Thus, for the sake of improving the image representation, it is possible to map multiple timestamps into one column or pixel. The number of timestamps present in one column or pixel is defined as ratio.

$$ratio = \frac{input\_width}{image\_width} \tag{6.2}$$

For example, whether the ratio is 2, each column of the image contains two timestamps $\{x_{t-1}, x_t\}$.

To generate the image representation of the series when the input_width ($\{x_0, x_1, \ldots, x_{input\_width}\}$) in longer than the image width, `matplotlib` is utilized. In this scenario, it is also possible to generate the image using either lines or dots. The result is presented in Fig. 31.

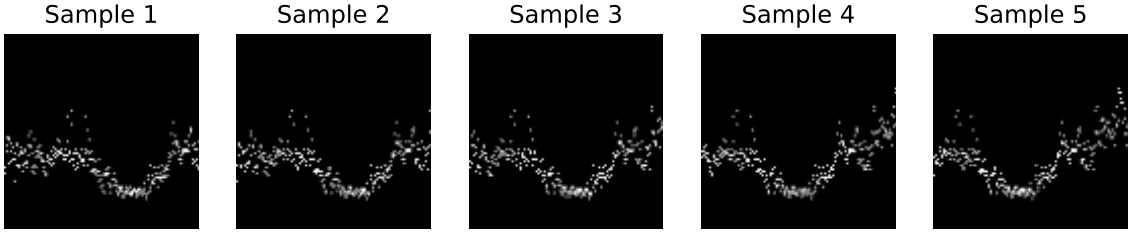


Figure 31. Example of image generated using matplotlib with ratio=3 (3 timestamps data points in one column image) .

When mapping from the image to numeric values back, the process is lossy, as there is no image column available for every timestamp presented in one image. Nevertheless, despite

this information loss, the model might perform better as now the main pattern could be represented in the image.

The performance for different ratios in the vanilla convolutional autoencoder is presented below.

Table 2. Prediction errors and performance metrics when changing the ratio between the input width and the image width.

ratio	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
1	80.832	**15.284**	103.852	1.365	0.098	39713.167	101.689	1.405	198.058
2	**77.957**	15.384	**99.551**	**1.317**	**0.102**	**38338.892**	97.779	**1.383**	198.058
3	107.558	19.866	134.712	1.816	0.084	49690.403	91.934	1.394	198.058
4	159.764	28.247	203.977	2.698	0.053	87348.006	94.731	1.406	198.058
5	166.033	28.773	206.901	2.804	0.055	127713.603	**55.148**	1.525	198.058

In Table 2 the metrics are presented when changing the ratio. The best metrics are found when the ratio is 2 (128 timestamps presented in 64 image width). These results are consistent with those presented in the previous section. When representing 128 timestamps, the pattern of the data is represented and the model performs best.

Image Size

For an image-based algorithm to work, it is essential that the image has a clear representation of the main pattern of the data. Whether the image shows no pattern and only randomness, the algorithm will only see noise as input and will output noise. In this context, increasing the width of the image (the height is always fixed to 100) could help show the pattern of the data in the image.

During this experiment, the ratio is always 1, when the input width is increased, the image width is increased in the same proportion.

Table 3. Prediction errors and performance metrics when changing the input width and the image width.

input	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
64	80.397	15.253	102.033	1.358	0.097	40031.059	102.171	**1.404**	**198.058**
100	997.846	216.875	1008.186	16.851	0.02	846173.478	**72.049**	1.417	287.223
128	**73.629**	**14.697**	**94.361**	**1.243**	**0.104**	**37157.107**	147.159	1.452	356.573
256	107.344	24.39	130.727	1.813	0.081	72722.405	238.488	1.585	673.603

Table 3 presents the different metrics when the input width and the image width are changed. Most metrics are observed to be better when the image width is 128. With this size, the image represents the pattern of the data for this specific dataset. The model size increases as we increase the image width, as the model has more parameters.

6.1.2 Image Dataset

Previous work in the field [16, 17], uses a synthetic dataset where the length of the series is less than 100 timestamps. Therefore, the full-time series could be represented in one 2D image. Nevertheless, this thesis uses a real dataset, where the length of the series is greater than 8000 timestamps. Thus, the whole cannot be represented in one image, and there is extra overhead in representing the series in multiple images.

To perform the data preparation, multiple images must be created to represent the entire time series (observe Fig. 32). From image to image, a shift of fh timestamps is created, where fh is the forecasting horizon. Thus, between two consecutive images, there is overlap. This overlap is defined as follows.

$$overlap = \frac{input\_width - forecasting\_horizon}{input\_width} \tag{6.3}$$

For example, whether the input_width is 64 timestamps and the forecasting horizon is 16, there is an overlapping of 75%.

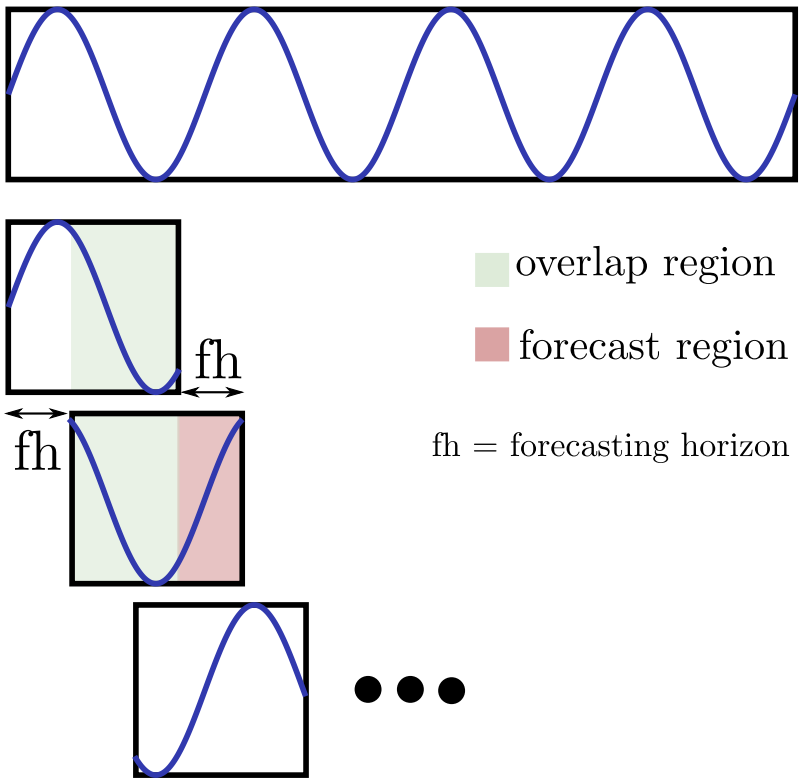


Figure 32. Illustration of the generation of a dataset with image sequences over synthetic data and 75% overlap between two consecutive images in the sequence.

Image-to-Image

Whether an image-to-image model is used, for each subset of the series $\{x_0, x_1, \ldots, x_t\}$, the input of the machine learning model is an image $\{x_0, x_1, \ldots, x_{input\_width}\} \mapsto I_t$. The labels or output is another image of the same size, but with a shift from the input of fh timestamps, as presented in Fig. 33

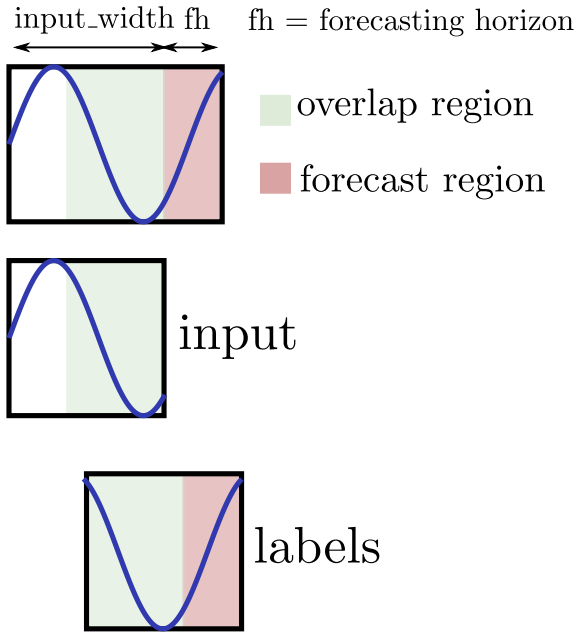


Figure 33. Input and labels for a training sample.

Image-to-Numeric

Whether an image-to-numeric approach is performed, the input of the machine learning model is the above-mentioned image($\{x_0, x_1, \ldots, x_{input\_width}\} \mapsto I_t$). The labels or outputs are the future timestamps of the series $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$.

6.1.3 Image Decomposition

The goal of a time series forecasting problem is to predict the values of the time series at future timestamps $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$ given historical data from the time series $\{x_0, x_1, \ldots, x_{input\_width}\}$. Whether an image-to-image approach is performed, it is necessary to map back the image predicted by the ML model to numeric values $I_t \mapsto \{x_0, x_1, \ldots, x_t\}$. This mapping is possible because the images have been generated with a specific methodology.

The process to map the image to numeric values is the following (observe Fig.34): first, the image is binarized so that there is only one white pixel per column of the image (i.e.,

numpy matrix). To binarize the image, the brightest pixel per column is extracted.

$$y\_value = argmax(column(image)) \tag{6.4}$$

where $y\_value \in [0, 100]$ is a value between 0 and 100 due to the height of the image being 100. Finally, the numeric value of the time series x_t is obtained by dividing y_value by 100:

$$x_t = y\_value/100 \tag{6.5}$$

where $x_t \in [0, 1]$. This process is repeated for every column of the image.

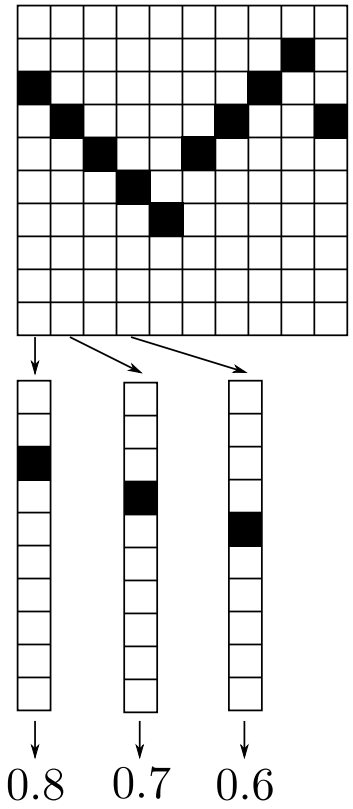


Figure 34. Illustration of the image decomposition process (transforming an image to numeric values).

When the input length is larger than the image width or the ratio is greater than 1, each column maps to $ratio$ number of data points. Thus, each column of the image corresponds to $ratio$ timestamps and the predicted value will be the same for each $ratio$ timestamps.

Fig. 35 shows an example of the input and labels sample for the training set, the output of the machine learning model and the image after binarization.

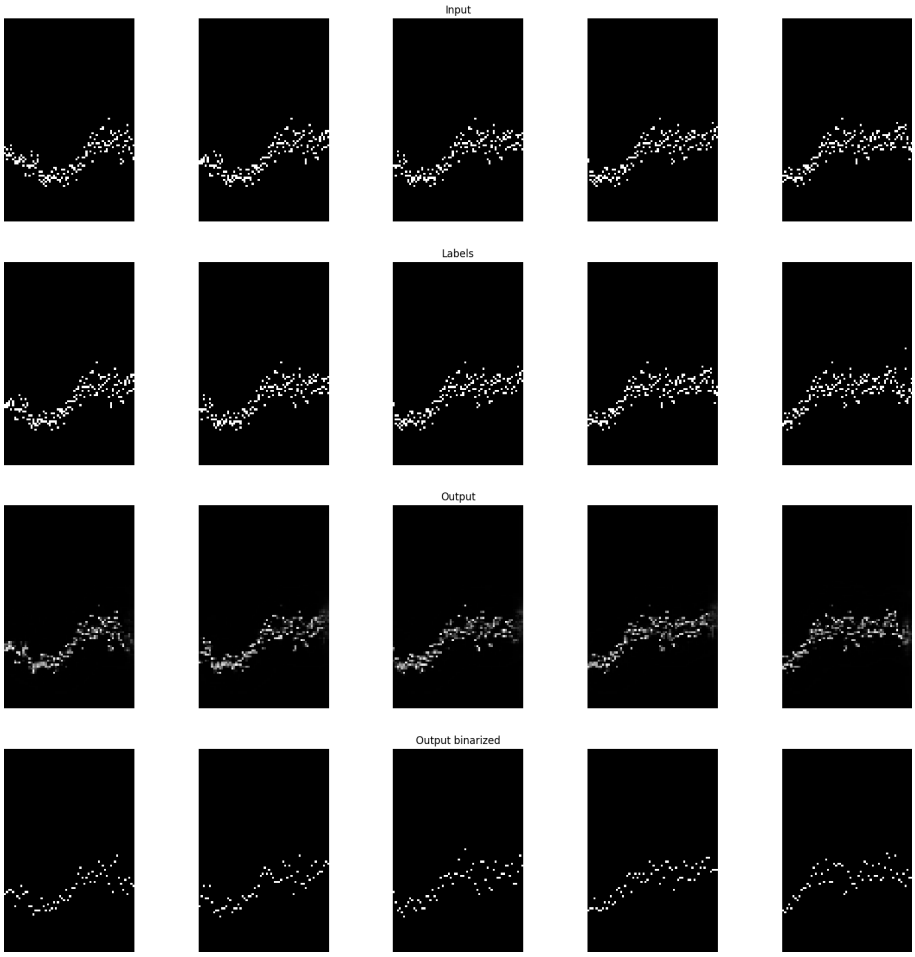


Figure 35. Example of the image decomposition process in one model. Input of the model. Labels of the model. Output of the model. Output of the model after binarizing the image.

6.2 Non-Image-based Machine Learning

When following a fully numeric procedure, data processing is also key to achieve good forecasting. To leverage the benefits of RNN, it is necessary to use window-based methods to generate sequence-to-sequence or sequence-to-one training samples.

Given a 1D time series of a random variable $\{x_0, x_1, \ldots, x_t\}$ where $x_t \in \mathbb{R}$, the goal is to predict the values of the random variables at future timestamps $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$ where fh is the forecasting horizon. To generate sequence-to-sequence or sequence-to-one samples, several timestamps are processed as input, that is, $\{x_0, x_1, \ldots, x_{input\_width}\}$ where $input\_width$ is the number of timestamps present in one sequence. And, the output or labels are processed as $\{x_{t+1}, x_{t+1}, \ldots, x_{t+fh}\}$. Whether the forecasting horizon is 1, a sequence-to-one problem is generated (observe Fig. 36. In other cases, a sequence-to-sequence problem is created (see Fig.37).

The forecasting problem could be formed as a regression problem whether the values are

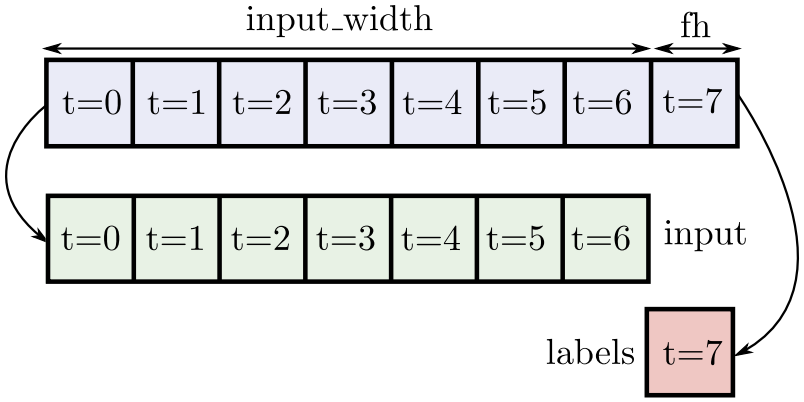


Figure 36. Illustration of Sequence-to-one data window split. Input and labels.

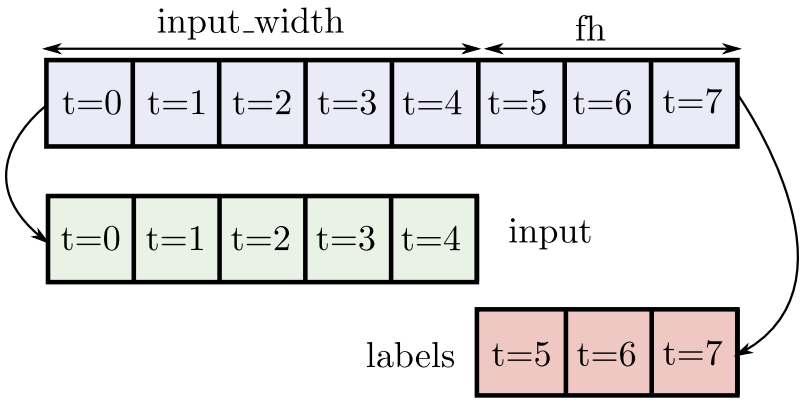


Figure 37. Illustration of Sequence-to-sequence data window split. Input and labels.

raw numeric values or as a classification problem if the raw numeric values are grouped in classes.

Regression

Whether a regression problem is performed, sequence-to-sequence or sequence-to-one training samples are generated. From one sample to the next, a shift of forecasting horizon (fh) timestamps is performed. Thus, each timestamp in the series is presented in the labels, and no data are skipped.

Classification

The regression problem could be transformed into a classification one. To achieve this goal, the raw numeric values are binned into discrete intervals (classes). The labels of these classes are the mean of the interval; thus, the classes could be mapped to the raw values afterward. Now, the raw numeric values belong to a certain class. The number of classes is determined by the number of bins created. In our experiments, it has been empirically impaired that 100 is a good number of bins.

In order to clarify the process, a simplified example is proposed. For example, whether the data contains values from 0 to 10 and the number and the number of bins is 2. The numeric values $x_t \in [0, 5)$ belong to the first class whose labels are the mean of the interval (i.e., 2.5) and the numeric values $x_t \in [5, 10)$ belong to the second class whose labels are 7.5.

It is noteworthy that this process is lossy. When mapping the numeric values to classes into bins, information is lost. Specifically, an error $\epsilon = \pm\,(bin\,interval\,length)/2$ is lost. There is a trade-off between the number of classes. Whether the number of classes is smaller, the machine learning algorithm usually performs better. On the other hand, having more classes or more bins yields a lower information loss when creating classes. As stated before, 100 classes or bins is a good empirical value.

Data pre-processing

The range of the raw data may vary widely across different datasets and objective functions will not work properly without a normalization step in the pre-processing. Furthermore, the gradient descend and other optimization algorithms converge much faster when feature scaling is performed.

In this work, a min-max normalization is performed for every machine learning algorithm. The scaling is performed using `MinMaxScaler` from `sklearn`. The process is defined by the following equation:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{6.6}$$

The parameters of the scaler are fit only in the training subset. Later, every subset is transformed using the parameters of the training subset.

Training Validation and Test Dataset

Commonly, in machine learning, train/test split splits the data randomly, since there is no dependence from one observation to another. However, this does not apply to time series data. In this type of data, there are dependencies between samples and one data point must follow its contiguous data point.

To properly validate the different models, 3 subsets are created: train, validation, and testing. The data comprise cloud resource utilization from 2013-08-12 13:40:00 to 2013-09-11 13:35:00, about a month of data. The splitting strategy is presented in Fig. 38:

- Training subset: 70%. [2013-08-12 13:40:00, 2013-09-2 13:05:00]
- Validation subset: 20%. [2013-09-2 13:05:00, 2013-09-8 13:45:00]
- Test subset: 10%. [2013-09-8 13:50:00, 2013-09-11 13:35:00]

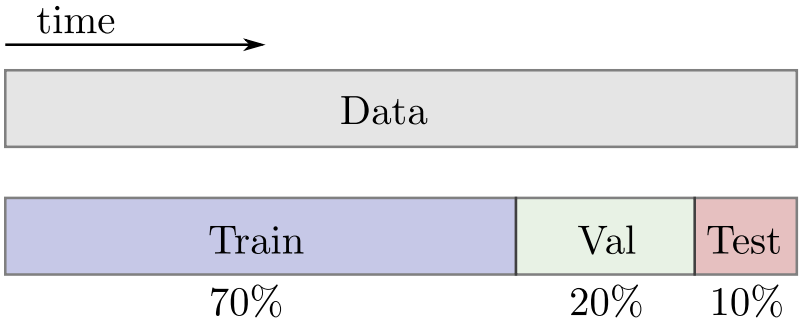


Figure 38. Illustration of Train/Validation/Test split.

Furthermore, during this thesis, another train/test split strategy has been explored. In order to analyze the performance of a model when it is trained in a certain dataset (VM) and inference in another dataset (other VM), the training and validation set correspond to one dataset and the test set is a completely new data set (another VM). In this scenario, the splitting strategy is the following:

- Training subset: 80% of VM1.
- Validation subset: 20% of VM1.
- Test subset: 100% of VM2.

7. Forecasting With Image-based Machine Learning Methods

Chapter 6 presented the advantages of using computer vision algorithms to forecast future cloud resource usage. In the mentioned chapter, it is also explained how to transform the numeric raw data into a 2D image representation and how to properly process the data to fit them into a deep learning model.

In this context, it is possible to use Autoencoder-based models where the input of the model and the output of the model is also an image (image-to-image) or other models where the input in an image but the output are numeric values (image-to-numeric). Autoencoders-based models have shown promise in the computer vision domain for tasks such as image denoising, image compression, and image completion and inpainting [55, 56, 57]. Furthermore, Autoeconders have also been used on numeric time series data with different variations (e.g., vanilla, convolutional, recurrent, etc.) for time series forecasting [58, 59, 60]. Using an image-to-image approach, it is possible to use fully computer vision approaches such as convolutional autoencoders or a combination of CNNs and RNNs as ConvLSTM [24] or LRCN [25]. The latter are commonly used in next-frame video prediction tasks [61].

From the previous chapter, it has been observed that **representing the pattern of the data in the image is the key** to enabling the machine learning algorithm to perform better. The different image generation techniques perform very similarly and thus **matplotlib with dots** is selected for its flexibility. Analyzing the results of the ratio and the input width, we observed that with **128 timestamps**, the model performs best. When having a ratio higher than 1, some information is lost when mapping one column to ratio timestamps. Thus, the chosen model is a **100x128 with a ratio of 1 and uses `matplotlib` with dots** to generate the image.

In this section, a comparison is presented between different image-based models. The different algorithms include image-to-image models (i.e., Autoencoder (see Section 4.4) and Video-frame prediction (see Section 4.4)) and image-to-numeric models (i.e., LRCN (see Section 4.4)). In this analysis, the effect of different parameters such as the forecasting history, the forecasting horizon, and the overlap in each model is presented. Later, a

comparison between the different models under the same conditions is presented.

7.1 Comparison of Forecasting History

The forecasting history or input length is defined as the number of timestamps present in an image, and thus the number of timestamps that feed it to the model. As mentioned in the previous section, the model needs that the input image contains the pattern of the data in order to learn it. The goal of this analysis is to observe how each model reacts when changing this forecasting history. During these experiments, the input length is the same as the image width, and thus, the ratio is always 1.

Convolutional Autoencoder

Table 4 shows the different metrics when varying the input length or forecasting history in the Autoencoder model. One could observe how the model performs better when having a forecasting history of 128 timestamps. When increasing up to 256 the history, the errors show that the model performs worse. Furthermore, when the forecasting history is increased, the training time and model size increase significantly in this model.

Table 4. Prediction errors and performance metrics when changing the forecasting history in the Autoencoder model.

input	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
64	80.397	15.253	102.033	1.358	0.097	40031.059	**102.171**	**1.404**	**198.058**
128	**73.629**	**14.697**	**94.361**	**1.243**	**0.104**	**37157.107**	147.159	1.452	356.573
256	107.344	24.39	130.727	1.813	0.081	72722.405	238.488	1.585	673.603

Video-frame prediction (ConvLSTM)

Table 5 shows the different metrics when varying the input length or forecasting history in the video-frame prediction model. The errors are slightly smaller when the input length is 64 timestamps. However, this model performs similarly for the different forecasting histories.

Table 5. Prediction errors and performance metrics when changing the forecasting history in the video-frame prediction model

input	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
64	**87.874**	**17.531**	**113.017**	**1.484**	**0.099**	**36679.399**	**119.031**	**2.13**	34.692
128	88.943	17.984	115.189	1.502	0.089	38182.025	148.189	2.184	34.692
256	93.011	18.879	119.093	1.571	0.091	37521.106	241.038	2.366	34.692

Long Recurrent Convolutional Network (LRCN)

Table 6 shows the different metrics when varying the input length or forecasting history in the LRCN model. The models perform similarly when the forecasting history is either 64

or 128 but get worse when increasing the input length to 256.

Table 6. Prediction errors and performance metrics when changing the forecasting history in the LCRN model.

input	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
64	71.806	14.360	91.507	1.213	0.118	39430.453	35.142	**0.921**	**0.954**
128	**69.825**	**14.106**	**89.812**	**1.179**	**0.125**	**38721.951**	28.387	0.932	1.052
256	124.607	28.724	148.979	2.104	0.051	104265.291	**24.542**	1.003	1.248

7.2 Comparison of Forecasting Horizon

The forecasting horizon is the number of future timestamps predicted by the model. The problem conditions the forecasting horizon. In some scenarios, it is necessary to forecast some seconds ahead, and in other scenarios, it is necessary to forecast one week ahead. In cloud environments, it is necessary to predict sufficiently in advance to be able to make the necessary changes to ensure uninterrupted service. This will depend on the resources of each data center, but cannot be too short.

Commonly, when using traditional methods to forecasting, increasing the forecasting horizon yields in a lower performance of the model (e.g, it is easier to forecast tomorrow's weather than next week's weather). Nonetheless, when using an image-driven approach, this result may be different and is explored in this section. During these experiments, the forecasting history is set to 128 timestamps. The forecasting horizon determines the shift between the training samples. Hence, when the forecasting horizon is larger, there are fewer training samples.

Convolutional Autoencoder

Table 7 shows the different metrics when varying the forecasting horizon in the Autoencoder model. As expected, the best performance is obtained when the model forecasts only one timestamp ahead. Nevertheless, the model performs better when the forecasting horizon is 16 than when it is 8. When increasing the forecasting horizon to 32, the model fails, and the errors are significantly higher.

Table 7. Prediction errors and performance metrics when changing the forecasting horizon in the Autoencoder model.

fh	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
1	66.692	13.147	86.409	1.126	0.124	37970.460	680.521	20.842	356.573
8	111.801	25.307	141.240	1.888	0.072	56808.144	184.491	2.823	356.573
16	73.956	14.706	96.657	1.249	0.110	36325.281	148.521	1.434	356.573
32	1000.762	218.591	1011.014	16.901	0.020	832633.724	65.420	0.850	356.573

Video-frame prediction (ConvLSTM)

Table 8 shows the different metrics when varying the forecasting horizon in the video-frame prediction model. This model decreases performance as long as the forecasting horizon is increased. However, the errors increase linearly, and the model still performs fairly when increasing the forecasting horizon.

Table 8. Prediction errors and performance metrics when changing the forecasting horizon in the Video-frame prediction model.

fh	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
1	62.041	12.241	81.295	1.048	0.151	13611.328	1532.751	22.625	34.692
8	78.138	15.327	101.380	1.320	0.127	30752.430	312.451	3.545	34.692
16	95.621	19.275	122.312	1.615	0.093	40815.739	159.111	2.214	34.692
32	113.451	21.713	146.505	1.916	0.076	50338.925	119.400	1.539	34.692

Long Recurrent Convolutional Network (LRCN)

Table 9 shows the different metrics when varying the forecasting horizon in the LRCN model. Again, as with the other models, the best performance is obtained when the model forecasts only one timestamp ahead. However, the model performs better when the forecasting horizon is 16 than when it is 8, as the Autoencoder model. When increasing the forecasting horizon to 32, the errors are higher, but fair.

Table 9. Prediction errors and performance metrics when changing the forecasting horizon in the LRCN model.

fh	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
1	60.838	12.091	78.794	1.027	0.144	35007.354	344.249	10.739	1.045
8	122.203	27.187	145.564	2.064	0.056	104063.931	38.403	1.592	1.048
16	70.800	14.494	90.603	1.196	0.126	39232.316	28.757	0.923	1.052
32	121.372	27.286	144.989	2.050	0.062	98994.833	15.037	0.671	1.058

7.3 Comparison of Overlap of the Input Image Sequence

As defined in Equation 6.1.2 in Chapter 6, the overlap is defined as the percentage of data shared between two consecutive training samples. The overlap between images is used to give the time notion to images, but it also serves as a sanity check on the effectiveness of a forecasting method. In the reconstructed overlap region, one could check whether the model is learning the pattern of the data. The overlap is directly related to the forecasting history and the forecasting horizon (see Equation 6.1.2). Thus, for this experiment, the forecasting horizon is set to 16, and as the overlap is changed, the forecasting history or input width is also modified.

Convolutional Autoencoder

Table 10 shows the different metrics when varying the overlap in the Autoencoder model. The model seems inflexible to variation in overlap. It performs significantly better when the overlap is 0.75 than in the other scenarios. We could observe that when the overlap is 50 % (see Fig. 39), there is not enough information in the image and the pattern is not shown. Thus, the output of the model is fuzzy, and the predictions are inaccurate. On the other hand, when the overlap is higher (90% in Fig. 40), a pattern is shown in the image, and the output of the model is significantly more precise.

Table 10. Prediction errors and performance metrics when changing the overlap in the Autoencoder model.

overlap	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
0	142.902	29.458	269.407	2.413	0.086	93860.918	61.282	1.333	79.171
0.5	802.686	166.771	852.201	13.556	0.014	671548.365	**47.143**	**1.426**	**118.800**
0.75	**81.994**	**15.544**	**104.606**	**1.385**	**0.099**	**40025.686**	101.365	1.380	198.058
0.8	138.827	24.535	174.627	2.344	0.065	100216.620	74.425	1.386	237.687
0.9	977.007	212.848	990.753	16.499	0.018	828501.805	94.888	1.537	435.831
0.95	792.415	166.905	844.226	13.382	0.012	656510.056	156.207	4.188	832.119

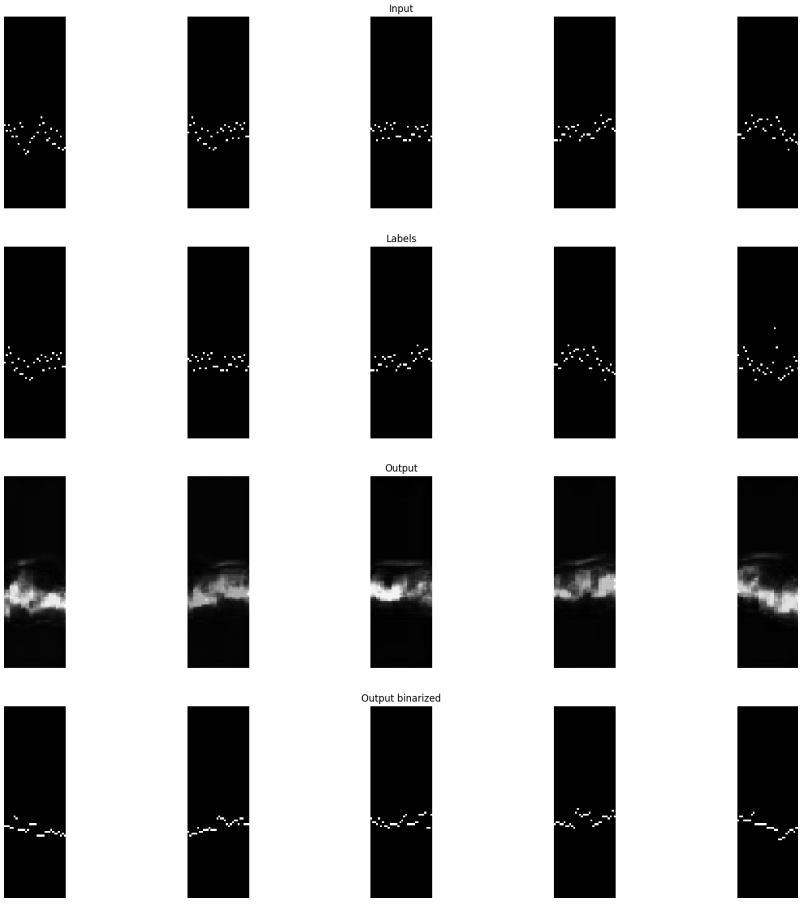


Figure 39. Output of the model when the overlap is 50%. The overlap is defined as the percentage of data shared between two consecutive training samples.

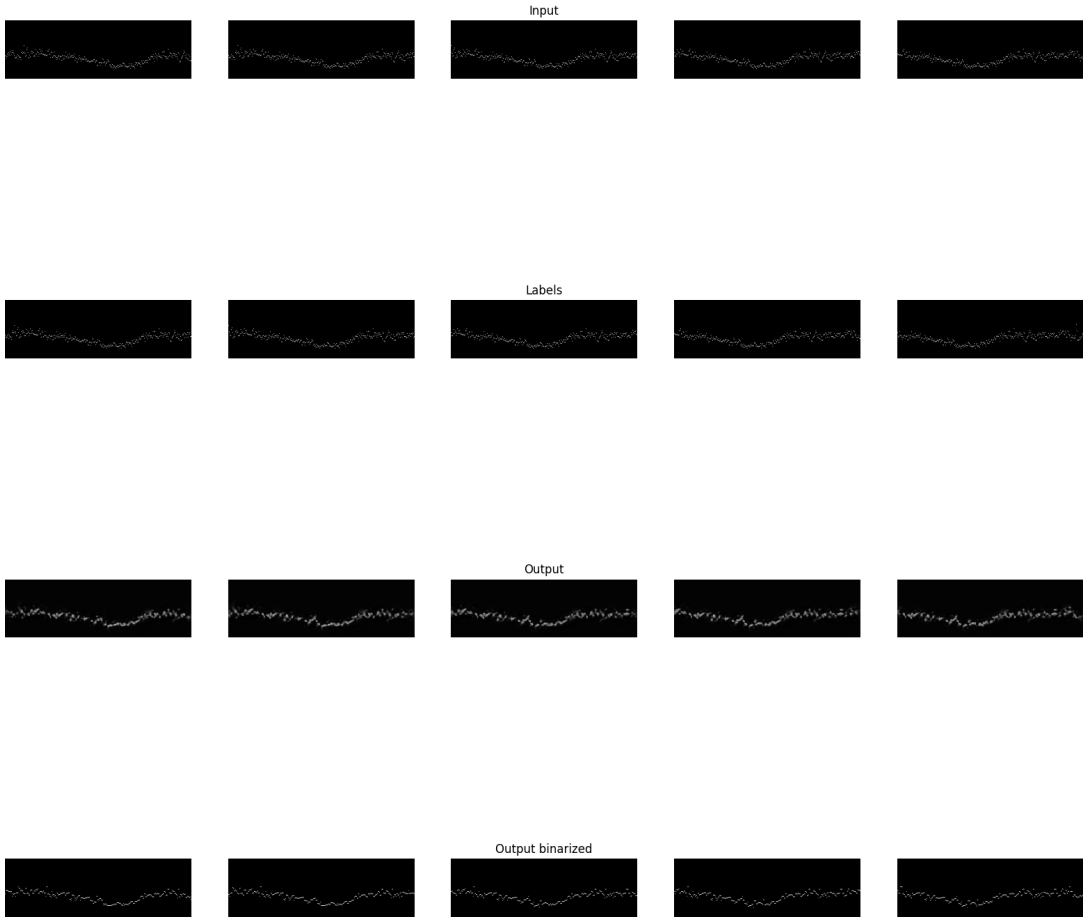


Figure 40. Output of the model when the overlap is 90%. The overlap is defined as the percentage of data shared between two consecutive training samples

Video-frame prediction (ConvLSTM)

Table 11 shows the different metrics when varying the overlap in the video-frame prediction model. In contrast to the Autoencoder model, this model is robust under variations in the overlap. The best performance is obtained when the overlap is 0.5, even though the variations are minimal.

Table 11. Prediction errors and performance metrics when changing the overlap in the video-frame prediction model.

overlap	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
0	88.956	17.931	111.852	1.502	0.079	35557.152	86.104	2.103	34.692
0.5	**83.160**	**16.484**	**108.285**	**1.404**	0.110	35897.407	**99.975**	**1.980**	34.692
0.75	87.675	17.380	112.429	1.481	0.094	39194.725	118.852	2.156	34.692
0.8	85.223	16.863	109.036	1.439	**0.114**	**33677.785**	131.855	2.033	34.692
0.9	89.630	18.392	112.480	1.514	0.078	34752.798	176.666	2.170	34.692
0.95	86.510	17.445	112.021	1.461	0.110	35266.127	291.069	2.445	34.692

Long Recurrent Convolutional Network (LRCN)

Table 12 shows the different metrics when varying the overlap in the LRCN model. This model is also robust under changes in the overlap. The best performance is obtained

when the overlap is 0.8, and the differences between models are slightly larger than the video-frame prediction model.

Table 12. Prediction errors and performance metrics when changing the overlap in the LRCN prediction model.

overlap	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
0	81.549	16.657	101.569	1.377	0.101	47266.186	16.238	0.428	0.972
0.5	76.772	15.363	97.468	1.297	0.106	42046.838	**30.344**	**0.991**	0.954
0.75	71.799	14.388	91.294	1.213	0.112	**39380.480**	38.375	0.937	0.954
0.8	**70.829**	**14.414**	**90.032**	**1.196**	**0.121**	39455.767	32.023	0.926	0.954
0.9	76.924	16.388	97.265	1.299	0.106	62800.633	41.883	0.948	1.052
0.95	122.380	27.451	146.018	2.067	0.064	100625.360	22.750	1.001	1.347

7.4 Performance Evaluation

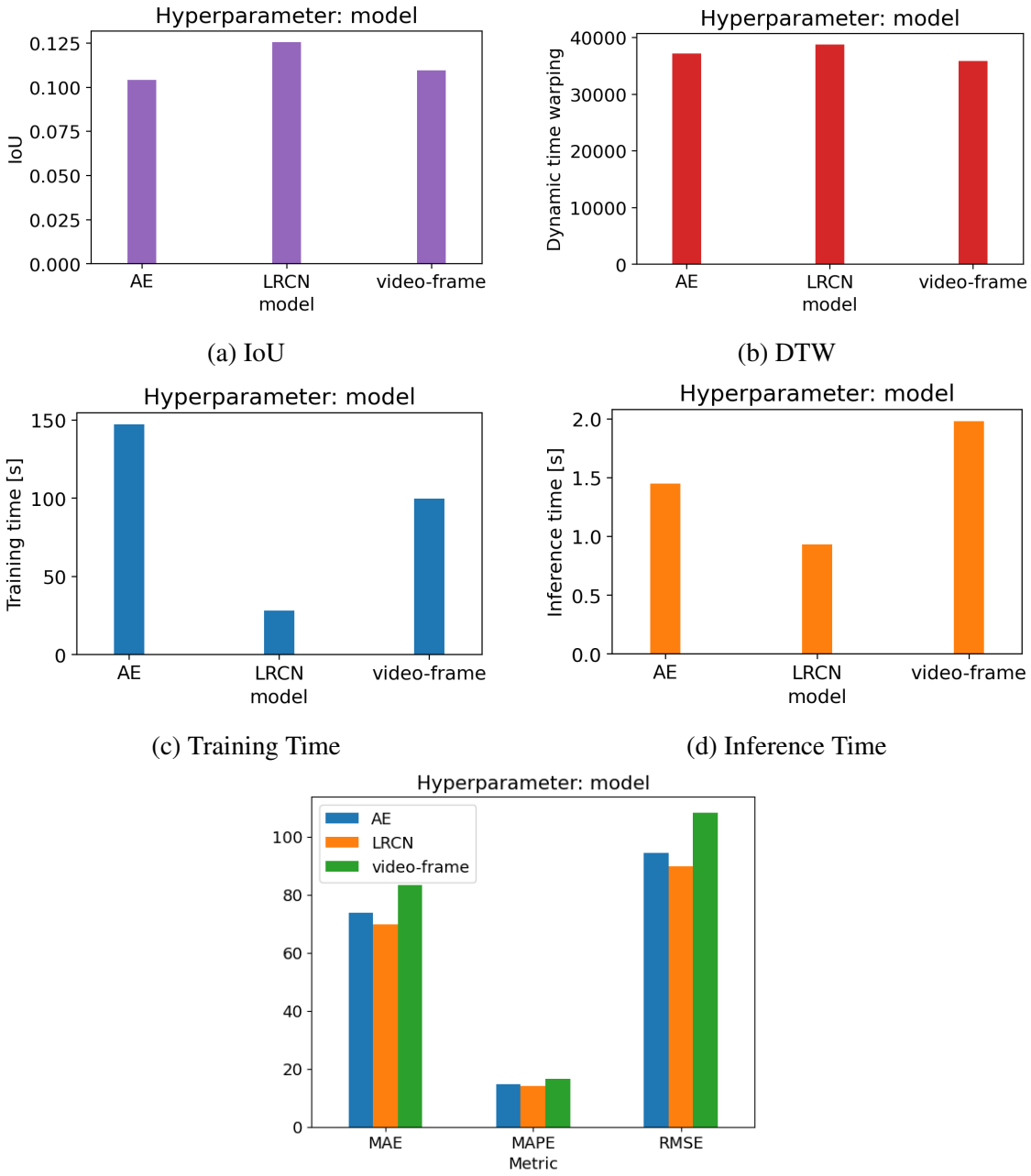
In the previous subsections, we have analyzed how the different parameters influence the models. Overall, it is observed that the video-frame prediction model is more robust under changes in hyperparameters than the Autoencoder and LRCN models. However, the performance in the video-frame prediction appears to lag behind the other models.

To better understand the differences between models, it is worth describing the difference between them. For this experiment, the best version of each model is selected for a fixed forecasting horizon of 16 timestamps. The images are 100x128 and thus the forecasting history is 128 and the overlap is 87.5 %. Table 13 reflects the differences in metrics for the different models. The autoencoder and the LRCN model outperform the video-frame prediction model. The LRCN model slightly outperforms the autoencoder model in terms of error. However, training times and model size are significantly smaller in the LRCN model compared to the other models. Fig. 41 shows these inputs graphically.

Table 13. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN and video-frame (ConvLSTM) models.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	73.629	14.697	94.361	1.243	0.104	37157.107	147.159	1.452	356.573
LRCN	**69.825**	**14.106**	**89.812**	**1.179**	**0.125**	38721.951	**28.387**	**0.932**	**1.052**
video-frame	83.160	16.484	108.285	1.404	0.110	**35897.407**	99.975	1.980	34.692

After analyzing the numeric metric, it is also noteworthy to visually inspect the differences between the model predictions. Fig. 42 shows how the ground truth and the prediction for the test subset for each model are. The LRCN model best captures the trend of the data (Fig. 42b). Furthermore, it also shows how the Autoencoder model is capable of capturing both long-term and short-term patterns. These insights are not reflected in a metric as a number but also define the performance of the different models. Being able to capture both long-term and short-term patterns may be very valuable in a cloud environment.



(a) IoU

(b) DTW

(c) Training Time

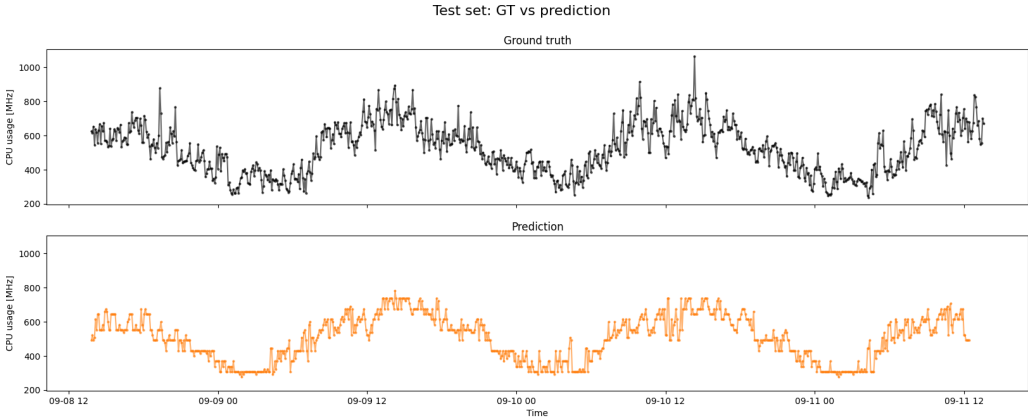
(d) Inference Time

(e) MAE, MAPE & RMSE

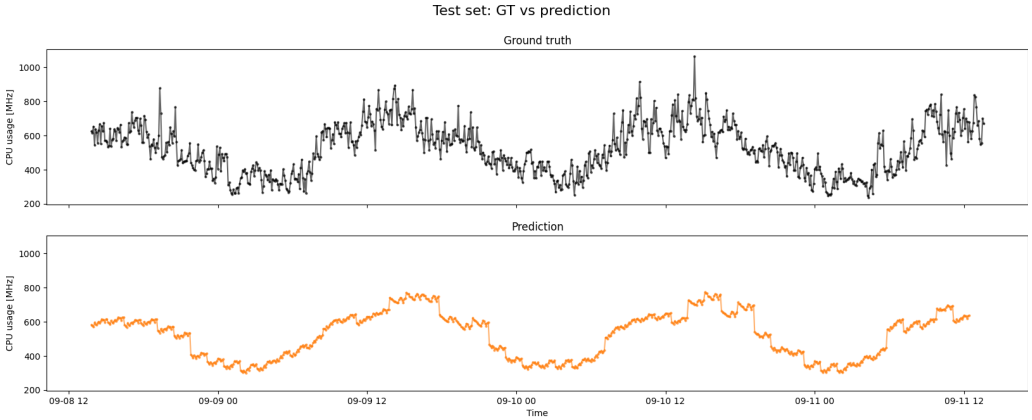
Figure 41. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN and video-frame (ConvLSTM) models.

7.5 Chapter Summary

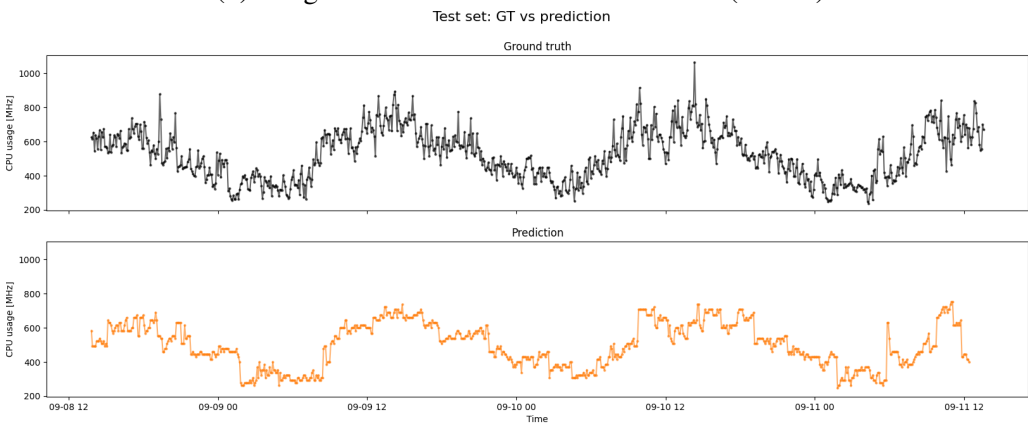
In this chapter, we have explored how to forecast future resource utilization by **leveraging computer vision algorithms**. We have shown how the **representation of the numeric data as an image is key** for the algorithm to work properly. Specifically, in a fully image approach where no RNN is included in the model, the performance of the model is disappointing whether the pattern is not shown in the data.



(a) Convolutional Autoencoder



(b) Long Recurrent Convolutional Network (LRCN)



(c) Video-frame.

Figure 42. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, LRCN and video-frame (ConvLSTM).

We have empirically observed that representing **128 timestamps** in each image yields the best performance of the model. 128 timestamps include data of approximately 12 hours or half a day. When showing less information in one image, the data in this image does not show any pattern and the model only see randomness. Whether the image is too wide, and thus the number of timestamps too large, the performance of the model is also reduced. 128 is also a good option, as it is a power of 2 and does not imply problems in some models

when downsampling.

For the sake of representing a pattern in a smaller image, it is possible to **represent more than 1 timestamp in each image column**. We have shown how this could help to improve the performance of the models. However, this process is "lossy" when mapping back the image to numeric. Hence, using larger images is preferred to representing more than one timestamp in one image column (having a ratio > 1).

As we have previously mentioned, 128 is a good forecasting history for the model. Furthermore, we observe how the **autoencoder** and the **LRCN** model achieve the best performance for a certain forecasting history, while the **video frame prediction model** is more robust and performs similarly for different forecasting histories. This is due to the fact that the **video prediction model** includes LSTM cells inside that capture the sequence information.

Commonly, in time series forecasting problems with traditional methods, the larger the **forecasting horizon**, the worse the performance (e.g, it is easier to forecast tomorrow's weather than next week's weather). Nonetheless, with an **image-driven approach this could differ**. We observe how the performance of the model does not decrease significantly as the forecast horizon increases. Additionally, in some models such as the autoencoder or the LRCN, the performance is better for a longer forecasting horizon.

The **overlap** determines the percentage of common data between consecutive images. However, it also determines the size of the image whether the forecasting horizon is set to a certain number. Now, we observe how the autoencoder model only performs properly for a certain overlap, while the video frame prediction model is very robust under changes in the overlap. The LRCN also performs properly for a wider range of overlaps.

Comparing the **best version of each model**, we observe how the three model perform similarly. The **LRCN** and the **autoencoder** slightly outperform the **video-frame** according to numeric errors. Nevertheless, by analyzing the predicted series of each model, we observe how the **autoencoder** is able to capture the short-term patterns of the data and the trend. The **LRCN** predicts the trend of the data very accurately and thus has low errors. The **video-frame predicition** model captures the short-term and the variance of the data, but does not predict the trend of the data accurately.

8. Comparison Against Other Forecasting Methods

In Chapter 7, an alternative for forecasting future resource utilization in cloud environments has been presented. This solution leverage the power of computer vision and deep learning to learn patterns in previous data and forecast future resource utilization.

In this chapter, we are going to compare the most important models of the image-driven approach (i.e., Autoencoder, video-frame and LRCN) with several baselines. The most widely used traditional forecasting techniques include exponential smoothing and ARIMA [10, 11]. Furthermore, in recent times, deep learning approaches have been applied in the domain of time series analysis, and RNN or LSTM in particular are widely used for time series forecasting.

Hence, the comparison would be between these different models:

- ARIMA
- Exponential Smoothing
- LSTM
- Autoencoder
- LRCN
- Video-frame prediction

For the sake of simplicity, the optimization of each model is not detailed in this thesis. For the baseline, the ARIMA model is optimized using auto arima from `pmdarima` [62]. The exponential smoothing model is used from the library `statsmodels` [63].

The LSTM model is implemented on Keras and Tensorflow. The optimization of this model is performed through trial-and-error hyperparameter tuning. The optimal hyperparameters are:

- Optimizer: Adam
- Regression problem
- # of layers: 1

- # of neurons: 20
- input length: 50

For image-driven methods, optimization is detailed in Chapter 7, the best model is utilized in each scenario.

It must be noted that while machine learning-based methods train for a longer time in a preproduction step, the inference is generally fast in the implemented model. These models have a defined architecture, so that during the training stage, the weights are optimized and saved. During the inference stage, the model has some specific weights and makes a prediction based on the input data. However, traditional methods such as ARIMA may train faster, but it is necessary to update the model in each iteration based on the past data during the inference stage. This could be very computationally expensive and may not be feasible in certain scenarios. This result is confirmed during the experiments in this chapter.

8.1 Forecasting Horizon

In this section, we compare the performance of the different models for different forecasting horizons. The forecasting horizon is the number of future timestamps predicted by the model. The problem conditions the forecasting horizon. In some scenarios, it is necessary to forecast some seconds ahead, and in other scenarios, it is necessary to forecast one week ahead. In cloud environments, it is necessary to predict sufficiently in advance to be able to make the necessary changes to ensure uninterrupted service. This will depend on the resources of each data center, but cannot be too short. A certain model might perform better for short-term predictions, whereas another might outperform for a longer forecasting horizon.

For these experiments, the data of the VM 917 (see Fig. 23) is utilized. The optimized version of each model is compared for different forecasting horizons.

Forecasting Horizon: 1 timestep

Table 14 and Fig. 43 compare the metrics for the different models when the forecasting horizon is 1. For such a small forecasting horizon, the traditional baseline outperforms the machine learning approach. The ARIMA model has the lowest numeric errors (i.e, MAE, MAPE, RMSE, MASE) (see Fig. 43e) and the highest IoU (see Fig. 43a). Only the DTW shows that the video-frame prediction model better captures the similarity between the ground truth and the prediction. This effect may be observed in Fig. 44 and Fig. 45.

Table 14. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when the forecasting horizon is 1 timestep.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	66.692	13.147	86.409	1.126	0.124	37970.460	680.521	20.842	356.573
ARIMA	**23.468**	**4.772**	**30.155**	**0.399**	**0.152**	19330.159	35.625	5821.502	nan
LRCN	60.838	12.091	78.794	1.027	0.144	35007.354	344.249	10.739	1.045
LSTM	49.731	9.608	65.691	0.840	**0.172**	21195.905	26.797	**0.244**	0.022
exp smooth	50.950	9.994	66.702	0.866	0.074	20705.896	**11.006**	11.006	nan
video-frame	62.041	12.241	81.295	1.048	0.151	**13611.328**	1532.751	22.625	34.692

However, in each iteration, the ARIMA model needs to update its parameters, and as the forecasting horizon is only one, this is extremely computationally expensive and time consuming, as shown in Fig. 43d. Other simpler models such as exponential smoothing might perform better, whether there are time or computation constraints.

These results are consistent with the usual statement in the machine learning community, for simpler problems, simpler models work best. The data have a clear pattern and the forecasting horizon is very reduced, thus the complexity of the forecasting problem is low.

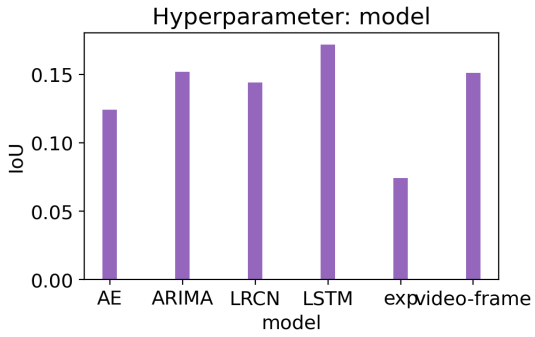
The autoencoder cannot learn the short-term pattern of the data (see Fig. 44a). And the LRCN model correctly learns the trend of the data, but again fails to capture the short-term patterns (see Fig. 44c).

Forecasting Horizon: 8 timesteps

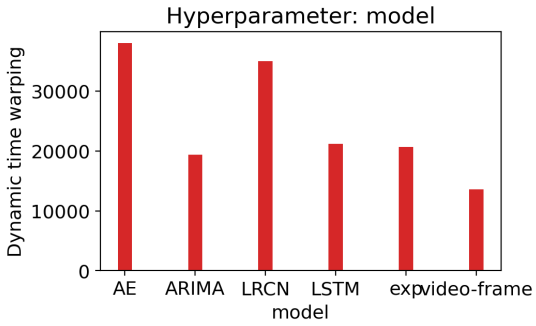
Table 15. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when the forecasting horizon is 8 timesteps.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	111.801	25.307	141.240	1.888	0.072	56808.144	184.491	2.823	356.573
ARIMA	**57.417**	**11.587**	**75.143**	**0.976**	0.039	36615.450	35.892	307.755	nan
LRCN	122.203	27.187	145.564	2.064	0.056	104063.931	38.403	1.592	1.048
LSTM	66.458	13.549	84.207	1.122	0.112	35974.762	6.423	0.237	0.023
exp smooth	65.141	12.877	84.755	1.107	0.045	35522.249	**1.397**	**1.397**	nan
video-frame	78.138	15.327	101.380	1.320	**0.127**	**30752.430**	312.451	3.545	34.692

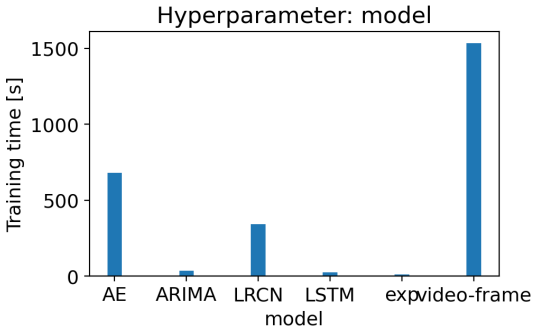
Table 15 and Fig. 46 compare the metrics for the different models when the forecasting horizon is 8. It is noteworthy the difference in performance according to different types of metrics. According to numeric errors (i.e., MAE, MAPE, RMSE, MASE), the arima model seems to outperform its competitors (see Fig. 46e). However, again, it is very computationally inefficient in the inference stage, as could be observed in Fig. 46d. Furthermore, the video-frame prediction model seems to outperform the other models in capturing both the short-term and the long-term patterns as proven by the IoU (Fig. 46a) and DTW (Fig. 46b) metrics. This effect is observed in Fig. 47 and Fig. 48, where traditional methods learn



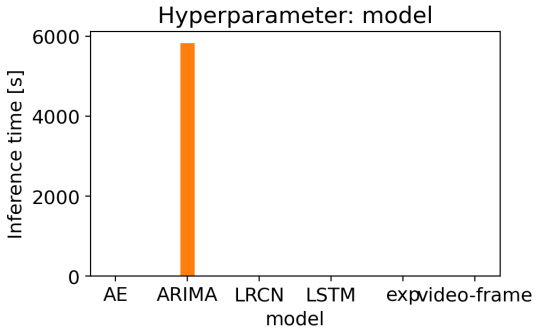
(a) IoU



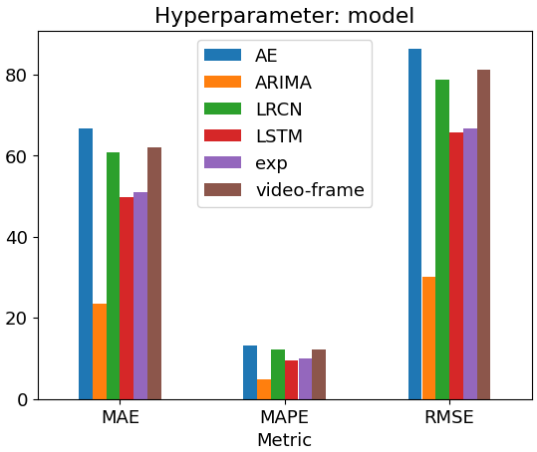
(b) DTW



(c) Training Time



(d) Inference Time

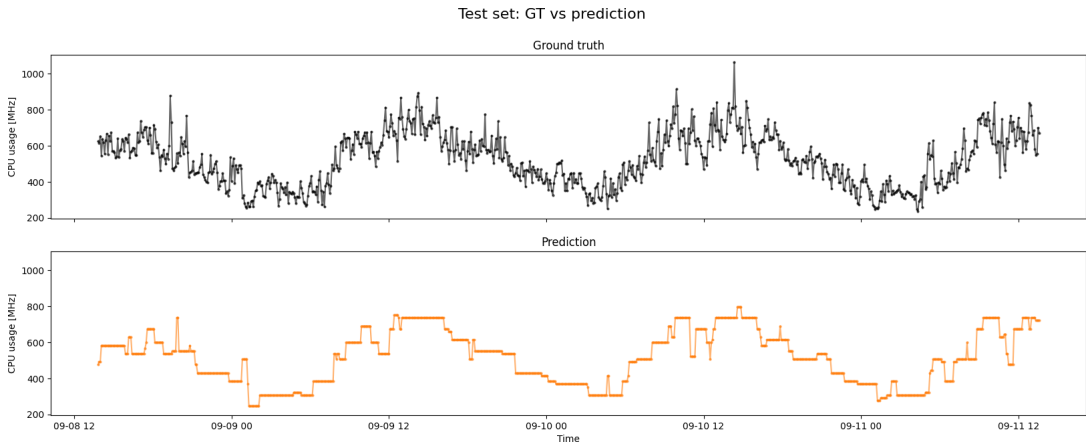


(e) MAE, MAPE & RMSE

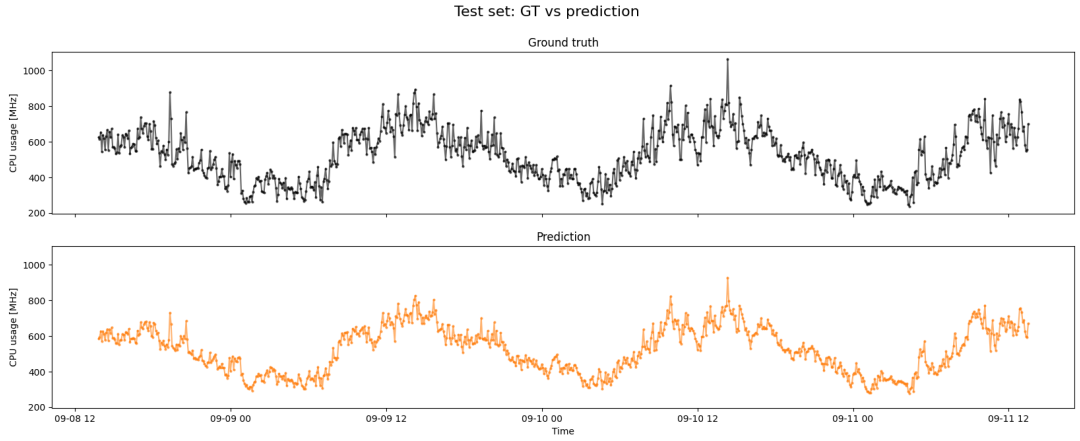
Figure 43. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models given a forecasting horizon of 1 timestep.

the trend of the data and the video-frame prediction model also captures the short-term patterns.

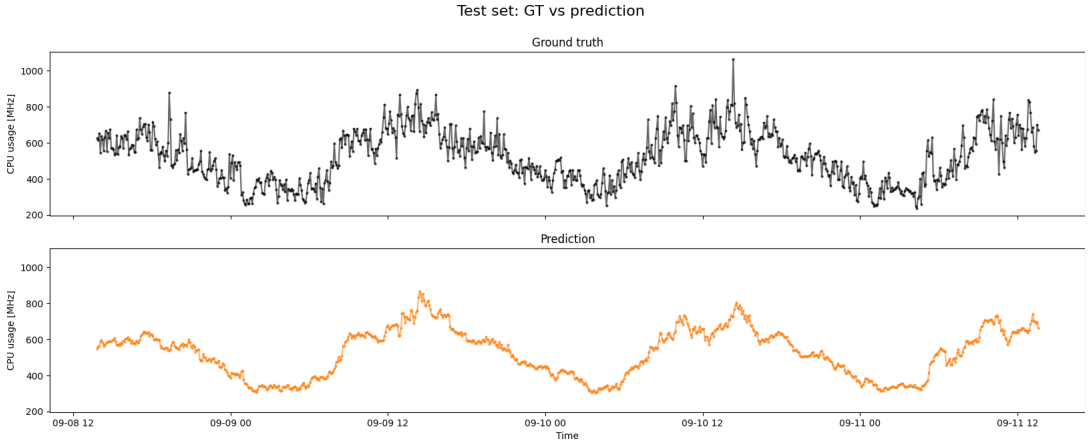
The autoencoder model captures the short-term pattern of the data but fails to correctly learn the trend of the data, as shown in Fig. 47a, so the numerical errors are higher. The LRCN model is not able to learn properly with this forecasting horizon and its predictions are very similar to the mean of the data (see Fig. 47c).



(a) Convolutional Autoencoder.



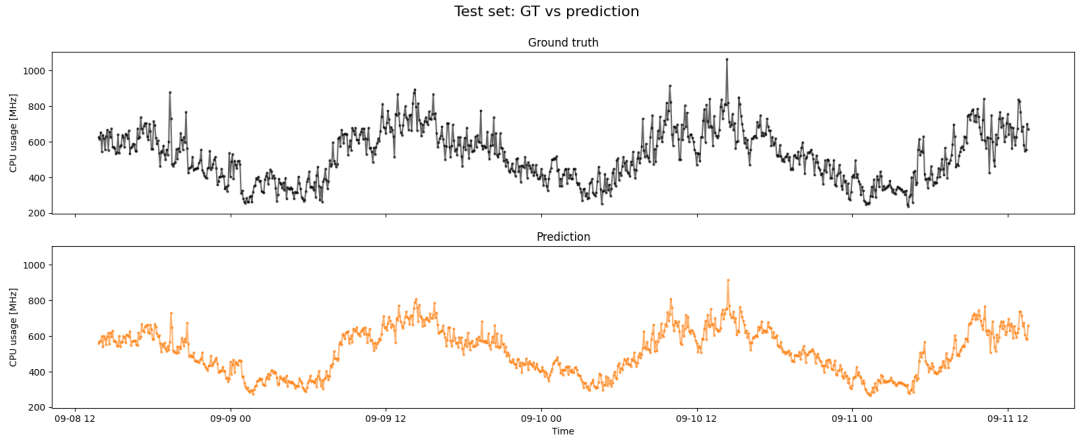
(b) ARIMA.



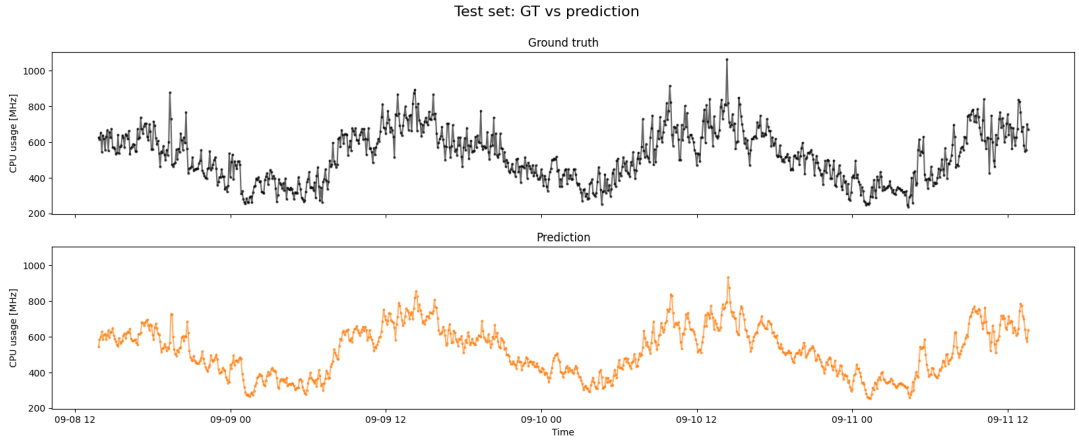
(c) Long Recurrent Convolutional Network (LRCN).

Figure 44. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, ARIMA, and LRCN given a forecasting horizon of 1 timestep.

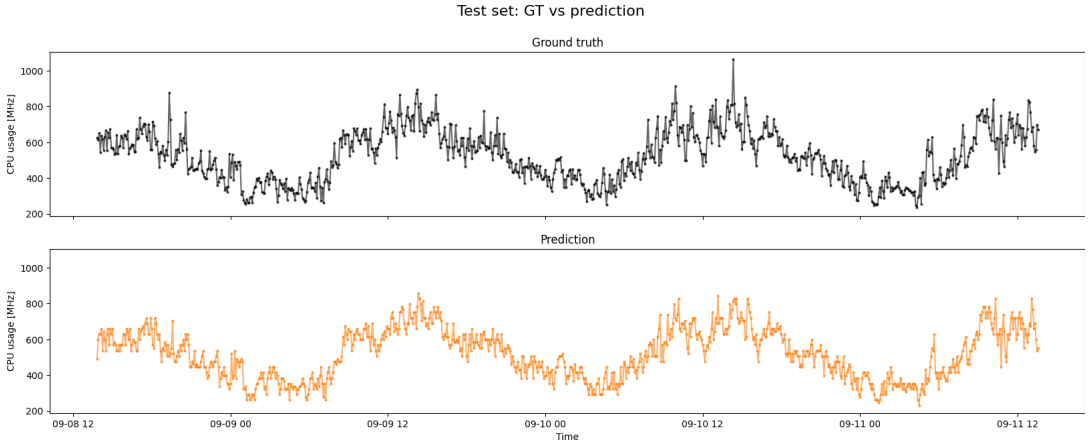
The LSTM model (see Fig. 48a) predicts the trend of the data with high precision. This yields a good performance according to the numeric errors and the IoU. However, the short-term pattern is not predicted and other models outperform this one according to the DTW metric..



(a) LSTM.



(b) Exponential smoothing.

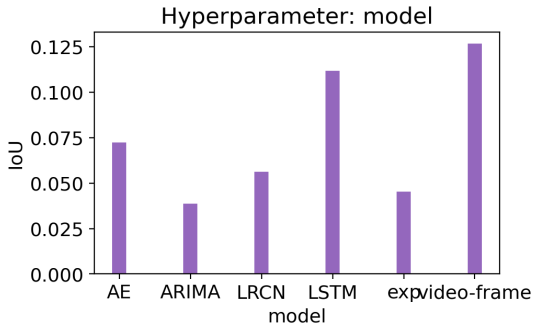


(c) Video-frame (ConvLSTM).

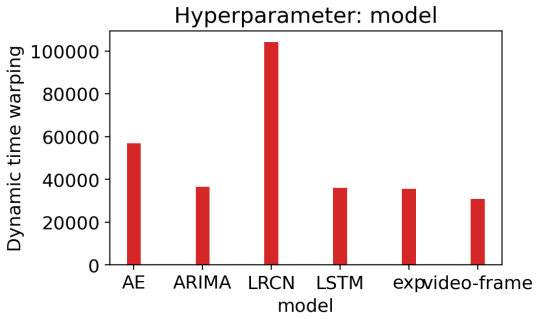
Figure 45. Comparison of real against the predicted CPU utilization by LSTM, Exponential Smoothing, and video-frame (ConvLSTM) given a forecasting horizon of 1 timestep.

Forecasting Horizon: 16 timesteps

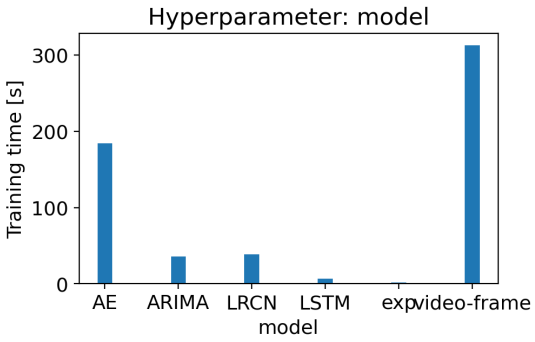
Table 16 and Fig. 49 compare the metrics for the different models when the forecasting horizon is 16. In this scenario, most of the models perform very similarly according to the numeric metrics (see Fig. 49e) and the DTW (see Fig. 49b). The ARIMA model



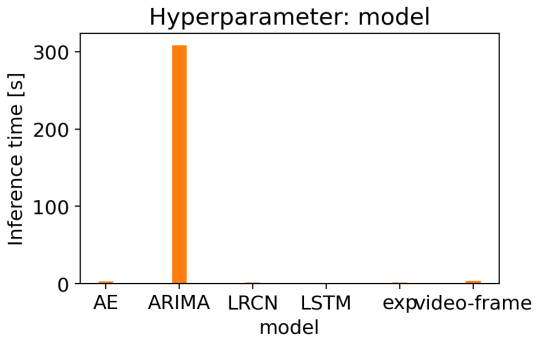
(a) IoU



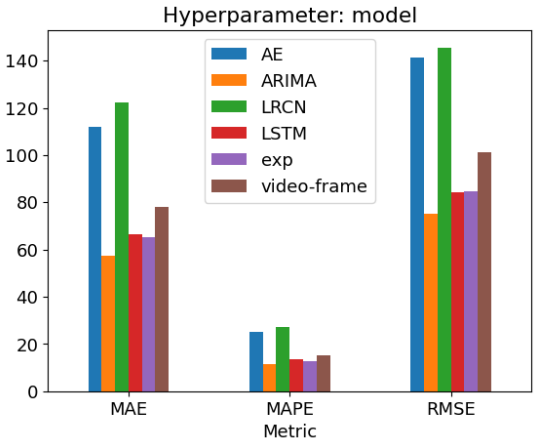
(b) DTW



(c) Training Time



(d) Inference Time

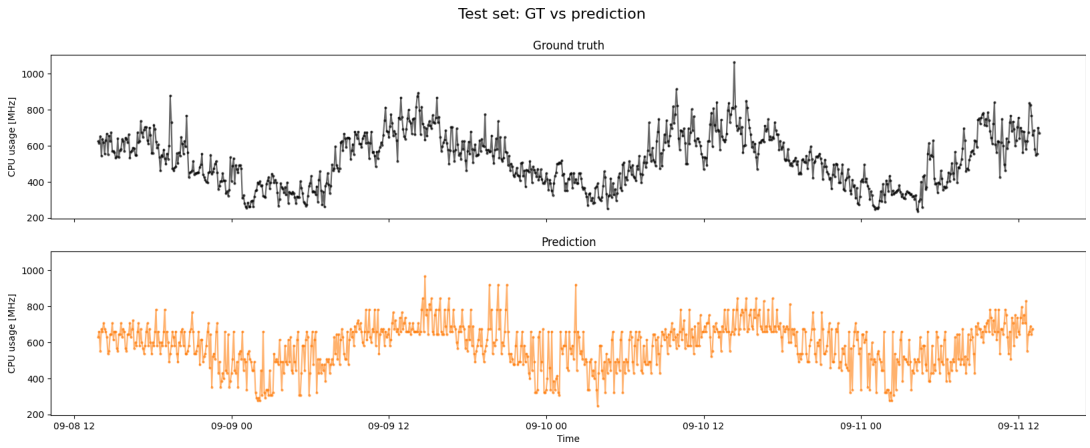


(e) MAE, MAPE & RMSE

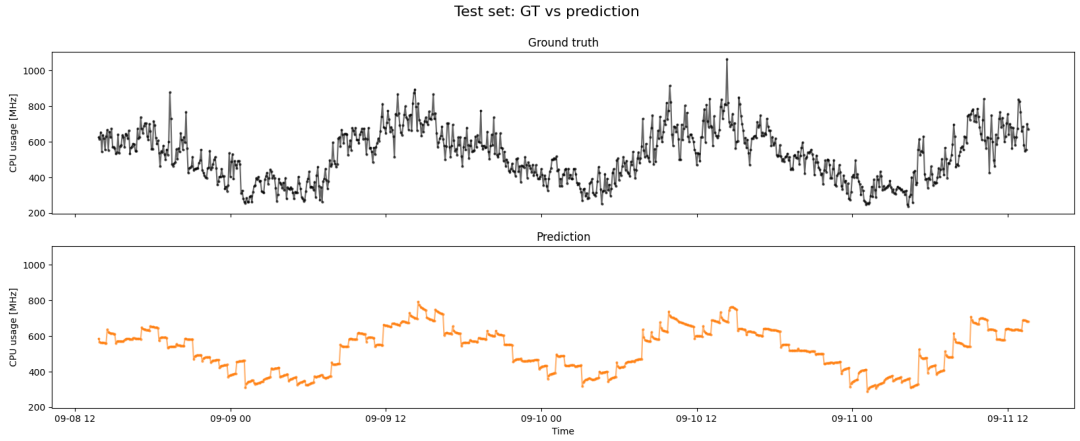
Figure 46. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models given a forecasting horizon of 8 timesteps.

slightly outperforms the other models in return of computational time in the inference stage (see Fig. 49d). Nevertheless, according to image metrics such as IoU (see Fig. 49a), the image-driven and the LSTM models outperform the traditional approach.

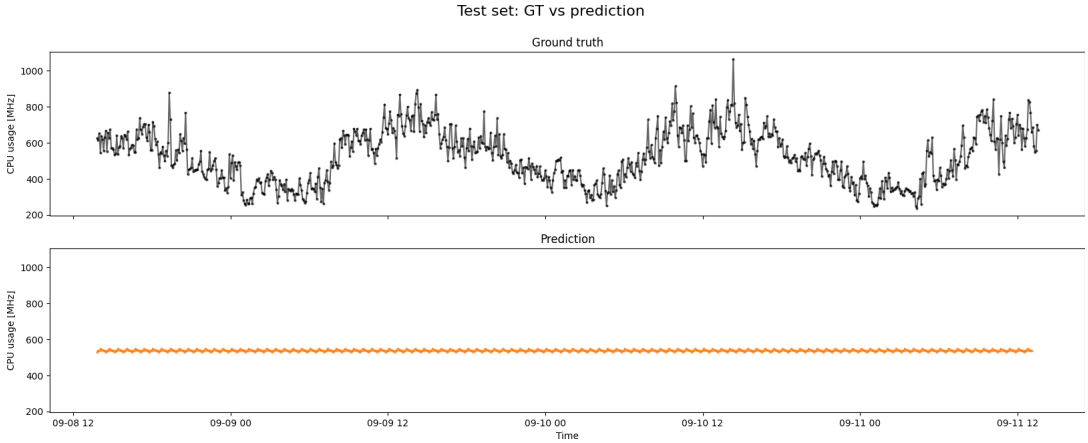
In Fig. 50b and Fig. 51b, we see how ARIMA and the exponential smoothing models, respectively, are able to follow the trend of the data, and this yields lower numeric errors but lacks in capturing the short-term pattern of the data that is reflected in lower IoU metrics.



(a) Convolutional Autoencoder.



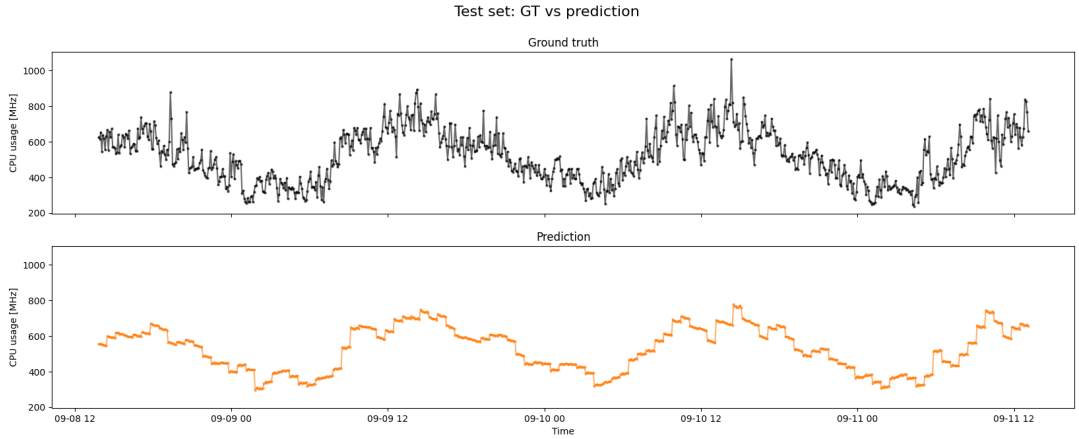
(b) ARIMA.



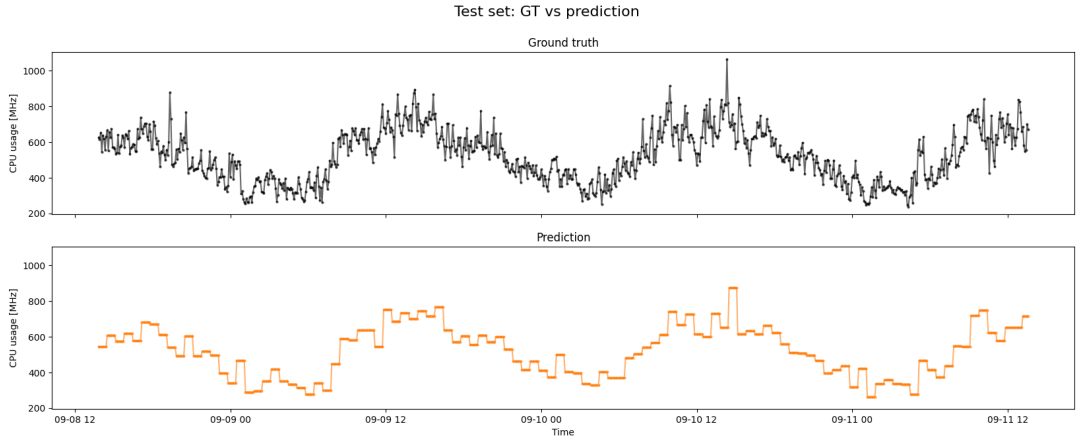
(c) Long Recurrent Convolutional Network (LRCN).

Figure 47. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, ARIMA, and LRCN given a forecasting horizon of 8 timesteps.

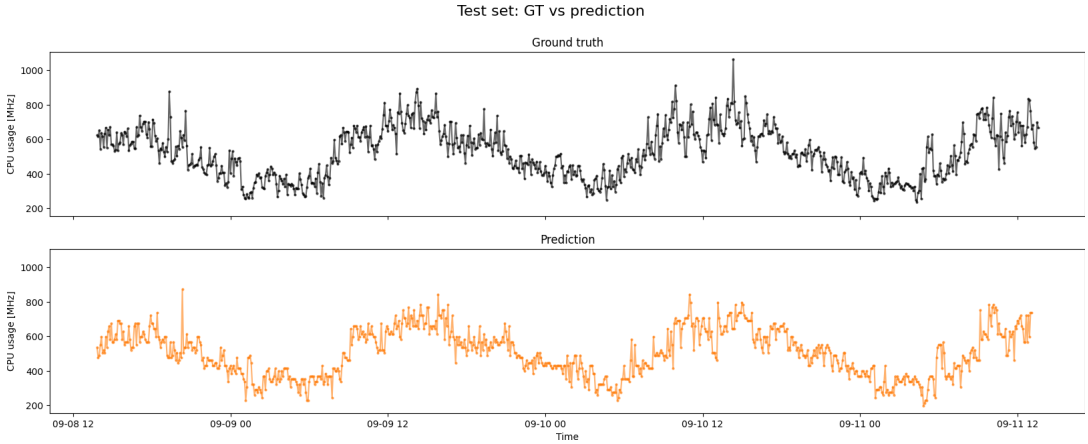
Between image-driven models, the autoencoder with the image-to-image approach better captures the short-term pattern of the data (see Fig. 50a). The image-to-numeric approach with LRCN (see Fig. 50c) and the numeric-to-numeric approach with LSTM (see Fig. 51a) accurately predict the trend of the data but do not capture the short-term pattern or the variance of the data.



(a) LSTM.



(b) Exponential smoothing.



(c) Video-frame (ConvLSTM).

Figure 48. Comparison of real against the predicted CPU utilization by LSTM, Exponential Smoothing, and video-frame (ConvLSTM) given a forecasting horizon of 8 timesteps.

Forecasting Horizon: 32 timesteps

Table 17 and Fig. 52 compare the metrics for the different models when the forecasting horizon is 32. Now, the complexity of the forecasting problem is greater.

Table 16. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when the forecasting horizon is 16 timesteps.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	73.956	14.706	96.657	1.249	0.110	**36325.281**	148.521	1.434	356.573
ARIMA	**68.819**	**14.018**	**89.550**	**1.169**	0.022	40134.383	35.633	147.211	nan
LRCN	70.800	14.494	90.603	1.196	**0.126**	39232.316	28.757	0.923	1.052
LSTM	74.397	14.734	95.181	1.256	0.108	39910.954	5.607	0.239	0.023
exp smooth	77.758	15.667	100.493	1.321	0.026	43220.586	**0.697**	**0.697**	nan
video-frame	95.621	19.275	122.312	1.615	0.093	40815.739	159.111	2.214	34.692

Table 17. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when the forecasting horizon is 32 timesteps.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	1000.762	218.591	1011.014	16.901	0.020	832633.724	65.420	0.850	356.573
ARIMA	**75.280**	**15.441**	**96.008**	**1.279**	0.033	**48176.562**	34.958	62.138	nan
LRCN	121.372	27.286	144.989	2.050	**0.062**	98994.833	15.037	0.671	1.058
LSTM	98.274	21.246	119.174	1.660	0.062	65289.742	5.401	0.241	0.025
exp smooth	82.316	16.538	104.764	1.399	0.014	53625.926	**0.357**	**0.357**	nan
video-frame	113.451	21.713	146.505	1.916	**0.076**	50338.925	119.400	1.539	34.692

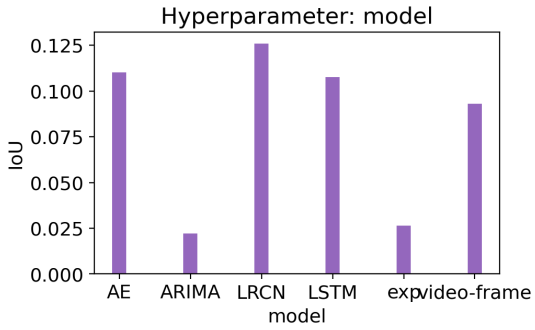
Models such as the autoencoder completely fail to forecasts future values or the LRCN always forecast the mean of the data, as could be observed in Fig. 53a and Fig. 53c, respectively.

Simpler models such as ARIMA or exponential smoothing perform better according to numeric errors (see Fig. 52e) but the trend or short-term pattern is not learned, as shown in Fig. 53b and Fig. 54b, respectively.

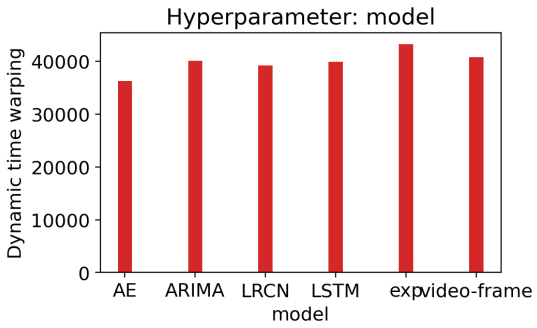
The video-frame prediction model is the one that best captures the trend and the short-term pattern of the data as reflected in Fig. 52a and could be observed in Fig. 54c.

Summary

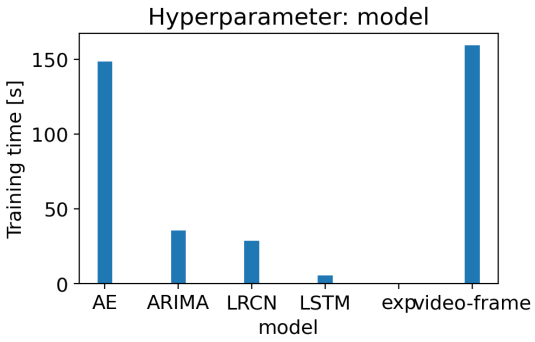
It is a very different problem to forecast tomorrow's weather than the next week's weather. A certain model might perform better for short-time forecasting periods and another one for longer forecasting periods. Usually, when the **forecasting horizon is very short (i.e., fh = 1)**, the model is simple and simpler models work best. This is observed during the experiments performed in this thesis. Simpler models such ARIMA or exponential smoothing work as well as a more complex model, and outperform a complex model in certain scenarios. However, it must be noted that while machine learning-based methods a train for longer time in a preproduction step, inference is generally fast in the implemented model. On the other hand, traditional methods such as ARIMA may train faster, but it is necessary to update the model in each iteration based on the past data during the



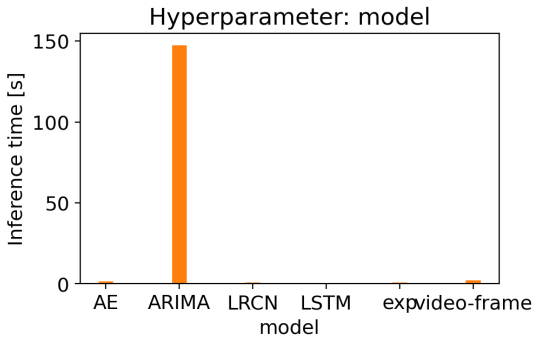
(a) IoU



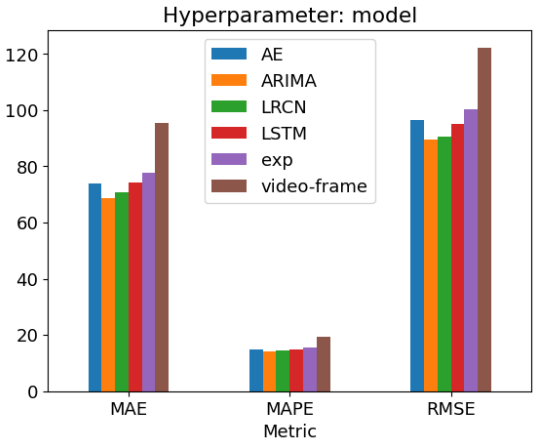
(b) DTW



(c) Training Time



(d) Inference Time

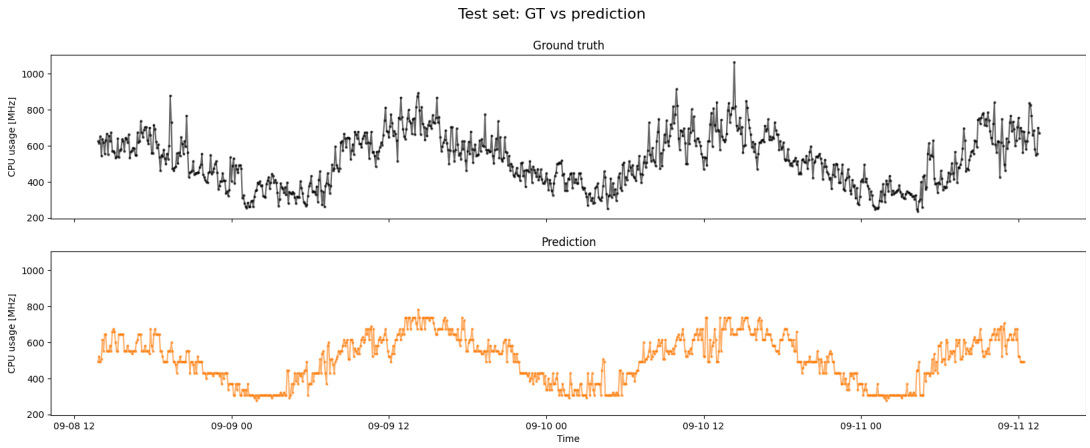


(e) MAE, MAPE & RMSE

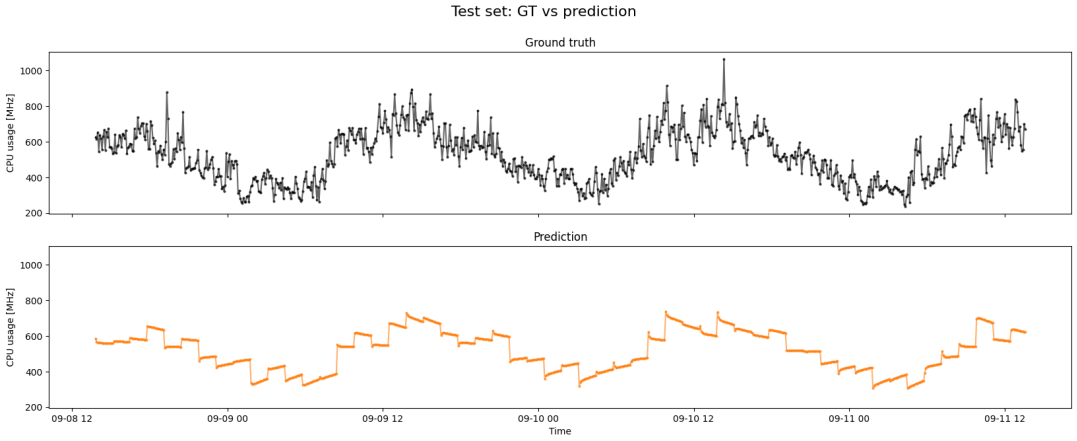
Figure 49. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models given a forecasting horizon of 16 timesteps.

inference stage. This could be observed during these experiments; ARIMA models usually accurately predict resource utilization, but updating the parameters, especially the moving average part, is very time-consuming and could be a significant disadvantage in certain scenarios, as may not be possible to implement due to time constraints.

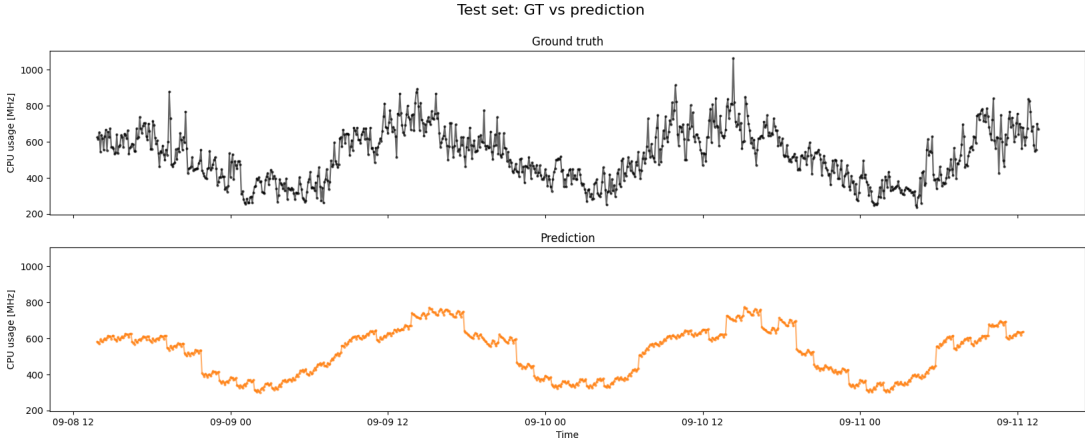
When increasing the **forecasting horizon (i.e., fh = 8)**, we see how the problem is more complex and the results are slightly different. The traditional baseline (i.e., ARIMA and exponential smoothing) is capable of predicting the trend of the data, but these models do



(a) Convolutional Autoencoder.



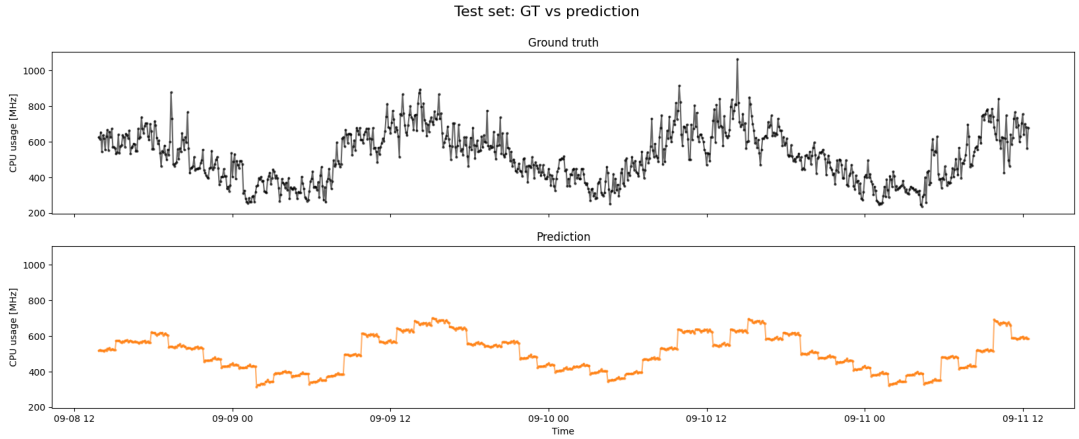
(b) ARIMA.



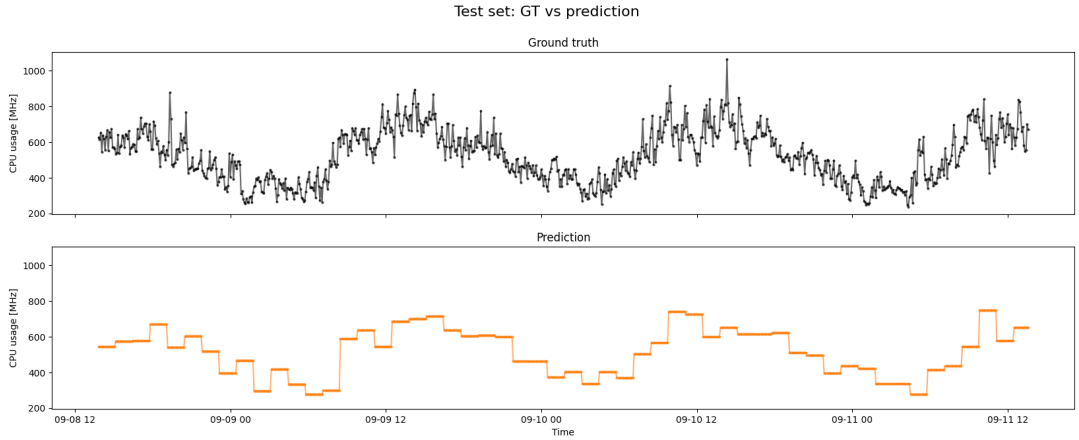
(c) Long Recurrent Convolutional Network (LRCN).

Figure 50. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, ARIMA, and LRCN given a forecasting horizon of 16 timesteps.

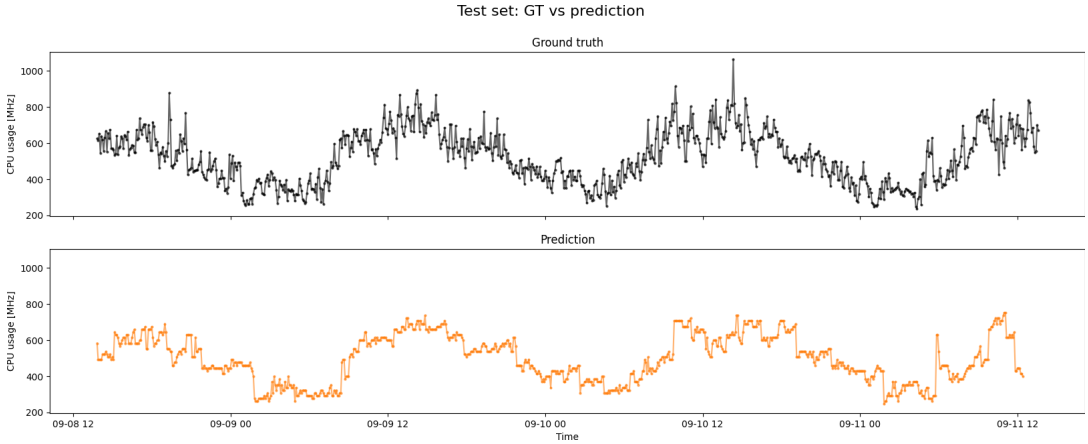
not capture the short-term patterns or the variance of the data. The video-frame prediction model best captures the short-term patterns and the variance of the data. For this forecasting horizon, the LSTM model predicts very accurately the trend of the data that yields good performance according to most metrics.



(a) LSTM.



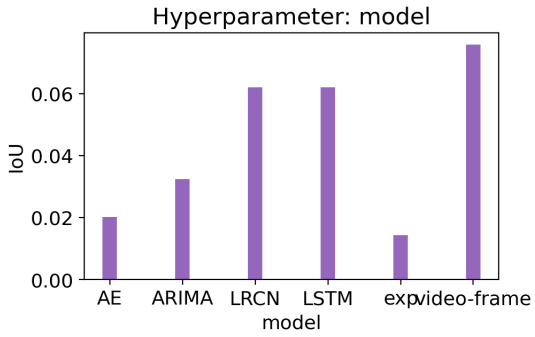
(b) Exponential smoothing.



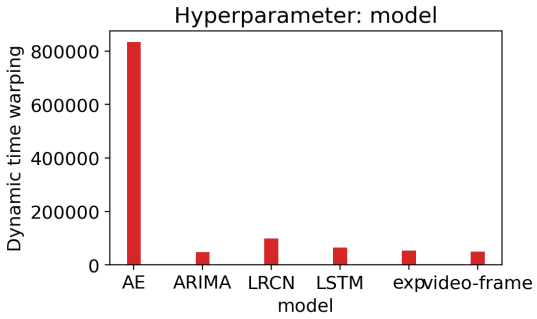
(c) Video-frame (ConvLSTM).

Figure 51. Comparison of real against the predicted CPU utilization by LSTM, Exponential Smoothing, and video-frame (ConvLSTM) given a forecasting horizon of 16 timesteps.

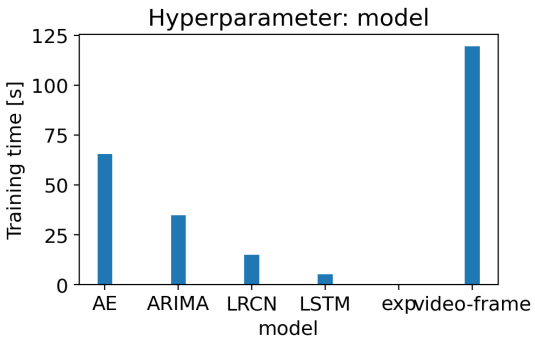
When the **forecasting horizon is even longer (i.e., fh = 16)**, the complexity of the problem reaches a point where the complexity of machine learning methods starts to be necessary. In this scenario, most algorithms perform similarly. The ARIMA, LRCN and LSTM models are able to predict the trend of the data accurately. However, only the autoencoder and the video-frame prediction models are capable of predicting the short-term pattern and



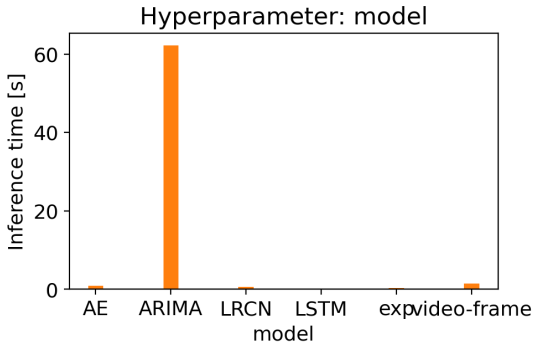
(a) IoU



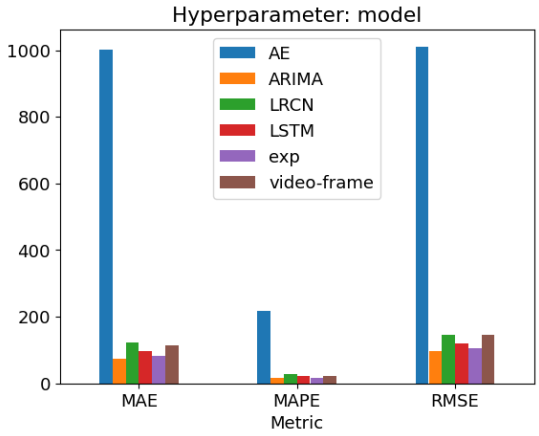
(b) DTW



(c) Training Time



(d) Inference Time

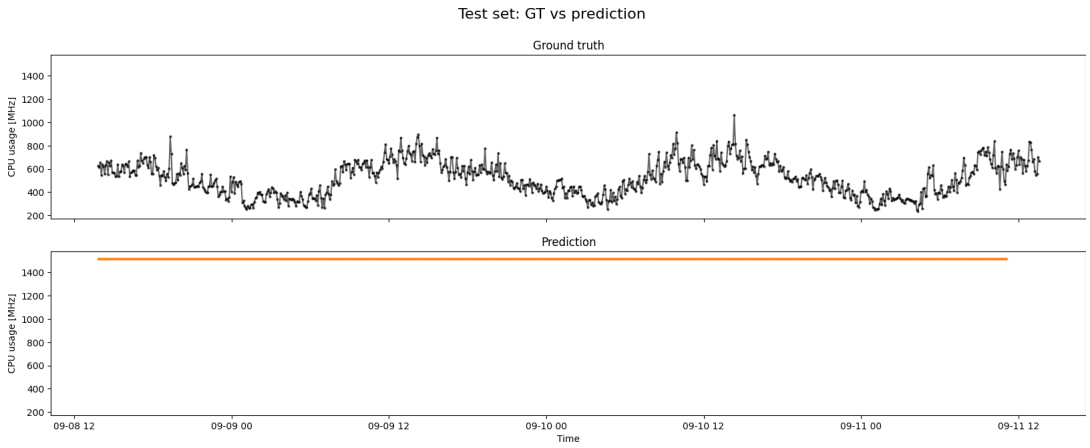


(e) MAE, MAPE & RMSE

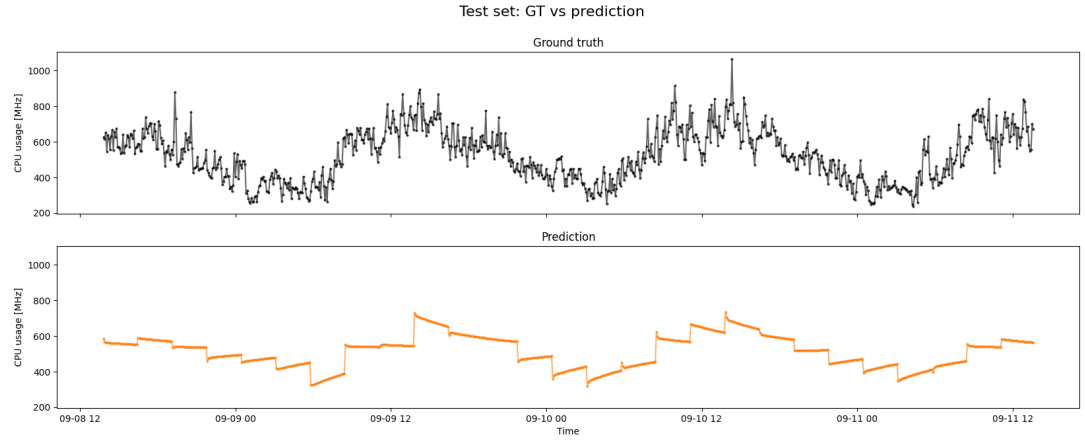
Figure 52. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models given a forecasting horizon of 32 timesteps.

the variance of the data.

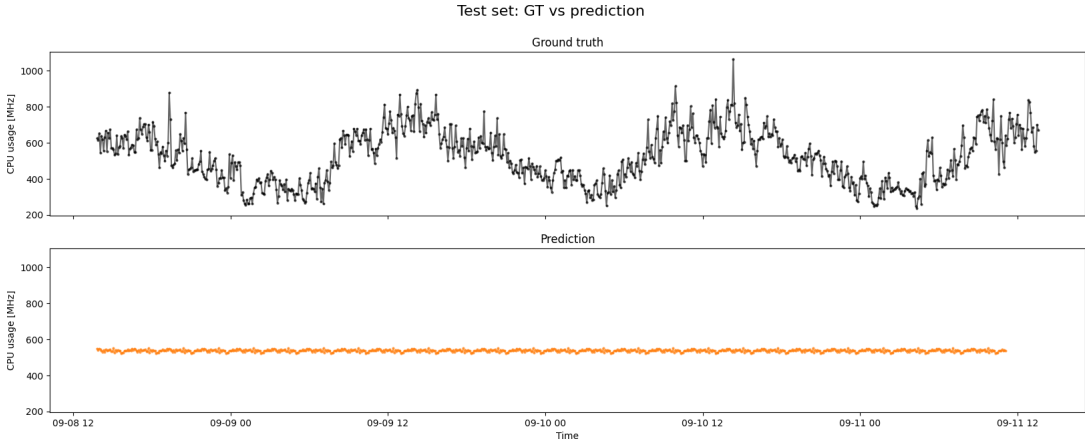
For a **very long forecasting horizon (i.e., fh = 32)**, the complexity of the problem is high and none of the model performs properly. The autoencoder and the LRCN completely fail to forecast the trend of the data. The ARIMA or exponential smoothing models also make a poor forecast of the data trend. The video-frame predicts a similar trend to the ground truth, but it is not accurate.



(a) Convolutional Autoencoder.



(b) ARIMA.

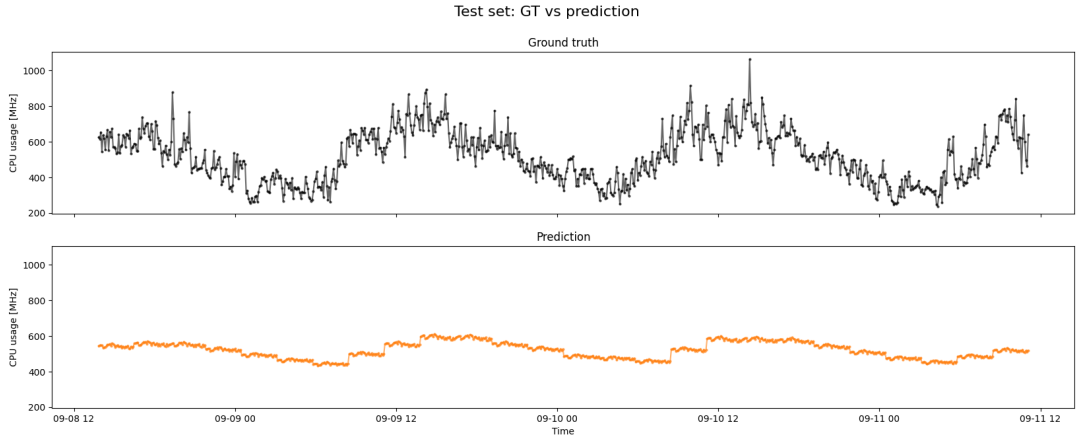


(c) Long Recurrent Convolutional Network (LRCN).

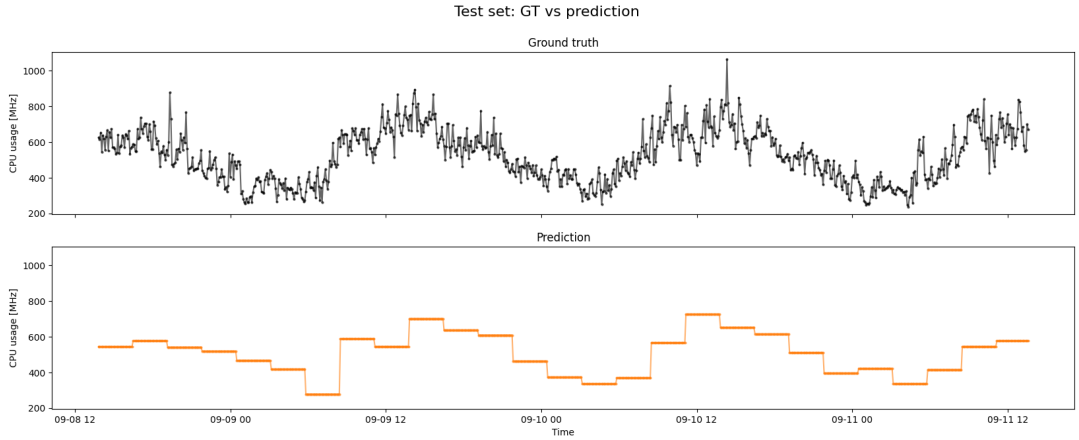
Figure 53. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, ARIMA, and LRCN given a forecasting horizon of 32 timesteps.

8.2 Inference on New Unseen Data

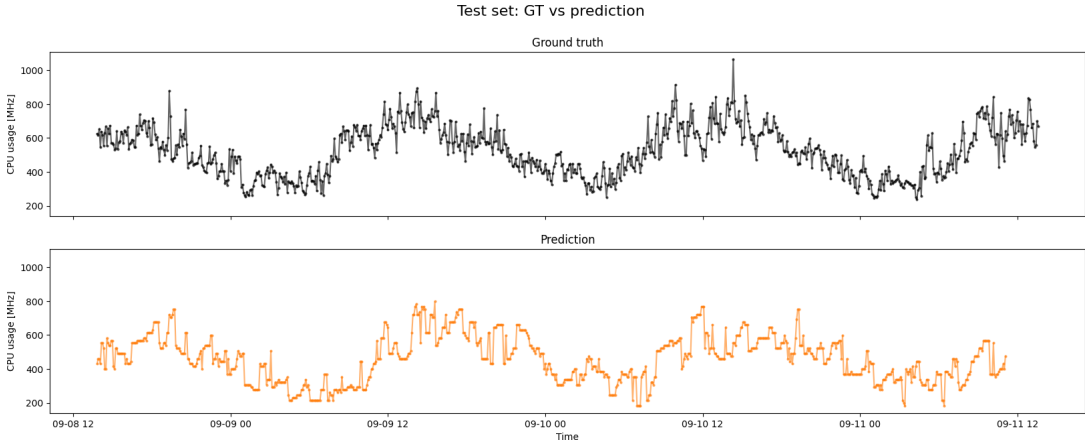
In this section, the performance of the different models is depicted for a new test set (i.e., a different VM of similar behavior). It is also important to understand the pattern of a VM and to be able to forecast the resource utilization of a fully unseen VM.



(a) LSTM.



(b) Exponential smoothing.



(c) Video-frame (ConvLSTM).

Figure 54. Comparison of real against the predicted CPU utilization by LSTM, Exponential Smoothing, and video-frame (ConvLSTM) given a forecasting horizon of 32 timesteps.

For these experiments, the train/val/test splitting is the following: the training and validation subsets are from VM 917 (see Fig. 23) and the test subset is another VM. The detailed procedure is explained in Chapter 4. The forecasting horizon is set to 16 timestamps.

This comparison is consistent only in the models that learn the patterns of the data in the

weights of the models and then make inferences based on these weights. Models such as ARIMA or exponential smoothing make their predictions based on previous data, updating their parameters in each iteration. However, exponential smoothing is also considered as reference. Nevertheless, it should be noted that this model predicts a weighted average of the previous data points for the next forecasting horizon (fh) data points. Thus, this could yield in predicting a trend that may have low errors in certain scenarios, but it would not forecast short-term patterns.

The goal of this experiment is to analyze how a model trained on a certain VM would work in another VM. The new unseen VM could have a similar pattern to the one that has been trained on, it could belong to the same cluster (clusters created in Chapter 5) or none of them. All these possibilities are explored in these experiments.

It is noteworthy that the MASE compares the performance of a certain model with the naive forecast in the training set. In this scenario, the training set and the test set may have different scales, and thus this metric is inconsistent for these experiments (see Chapter 4).

8.2.1 Same Cluster, Similar Pattern

Table 18. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, Arima, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the same cluster and similar pattern.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	247.366	75.593	285.351	3.086	0.024	1145966.327	162.497	12.994	356.573
LRCN	198.016	60.156	219.212	2.470	0.020	971291.960	33.122	6.658	1.052
LSTM	105.946	33.783	126.464	1.322	0.075	673848.451	**14.103**	**0.323**	**0.023**
video-frame	**82.575**	**24.288**	**111.496**	1.030	**0.145**	**362447.247**	160.814	14.351	34.692
exp smooth	67.326	18.696	96.109	1.139	0.057	380022.593	9.054	9.054	nan

Table 18 and Fig. 55 show the metrics for the different image-driven models when inferred in VM 340 (see Fig. 24). The VM 340 has a "sinusoidal" shape, similar to the training VM. Furthermore, it belongs to the same cluster according to the one created in Chapter 5, thus it is considered as having similar shape and belong to the same cluster as the training VM (VM 917).

The best performance is obtained for the video-frame prediction model for every metric. The video-frame is able to learn the patterns of one VM and forecast in a different VM and significantly outperforms the other models. The video-frame prediction model outperforms LSTM by 21% according to numeric errors (MAE) and has 93% higher IoU. Compared to the LRCN model, the numeric error (MAE) is 58% smaller and the IoU is 6 times bigger. The LRCN model has lower training time, inference time and model size, but the errors

are significantly higher.

In Fig. 57b could be observed how the video-frame model is capable of learning the trend, but also the short-term pattern of the data. The LRCN model is capable of learning the trend of the data, but is not able to predict the short-term patterns (see Fig. 56b). The autoencoder prediction is not accurate and seems fuzzy (see Fig. 56a).

The LSTM model is also capable of learning the trend of the data (see Fig. 56c). However, the model does not capture the variance of the data or the short-term patterns.

As a reference, the exponential smoothing model is presented (see Fig. 57a). As it is a weighted mean of previous timestamps, the data trend is captured, and this yields low errors. Nevertheless, once again, this model is not learning from previous data, and thus the variance or the short-term pattern of the data is not predicted.

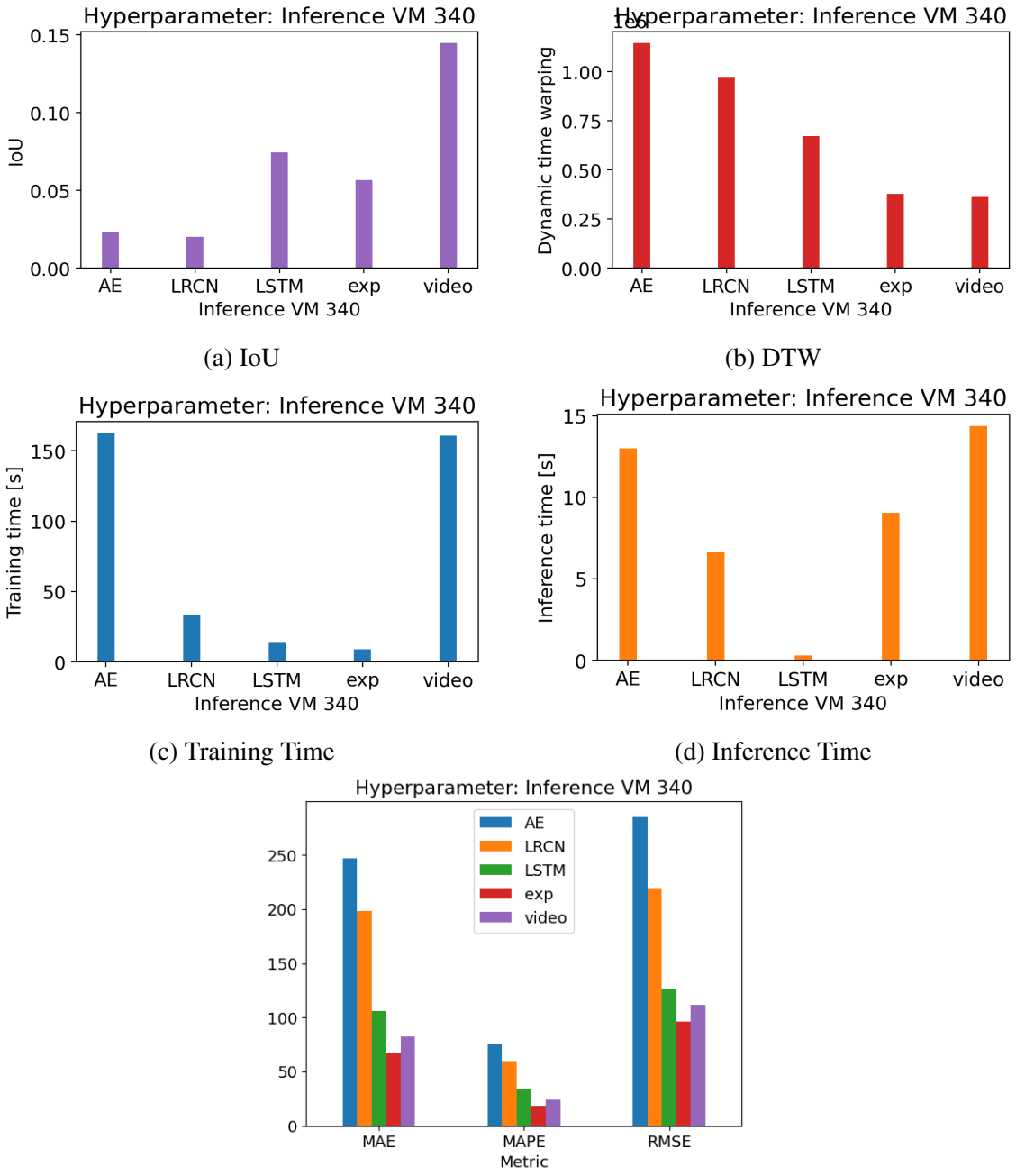
8.2.2 Same Cluster, Different Pattern

Table 19. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the same cluster and different pattern.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	3759.449	49.534	4908.521	7.980	0.034	25385972.637	161.842	12.899	356.573
LRCN	2307.046	27.984	2675.931	4.897	0.014	17616358.736	33.620	6.694	1.052
LSTM	1175.692	14.532	1538.917	2.496	0.030	7521483.200	**5.818**	**0.261**	**0.023**
video-frame	**952.432**	**11.466**	**1634.459**	2.022	**0.135**	**3576969.521**	158.336	14.477	34.692
exp smooth	679.599	7.805	1378.661	11.496	0.010	3125323.409	9.073	9.073	nan

Table 19 and Fig. 59 show the metrics for the different image-driven models when inferred in VM 119 (see Fig. 58). The VM 119 does not have a "sinusoidal" shape, like the training VM. However, it belongs to the same cluster according to the one created in Chapter 5, thus it is considered to have a different shape but belongs to the same cluster as the training VM (VM 917). In this experiment, we will show how the model is capable of forecasting in a new VM with different behavior.

The video-frame prediction model outperforms the other models according to numeric metrics (see Fig. 59e), IoU (see Fig. 59a) and DTW (see Fig. 59b). As shown in Fig. 61b, this model predicts the trend of the data and also the short-term pattern, the variance of the data is slightly over-estimated. The video-frame model outperforms the next best competitor (LSTM) by 19% according to numeric errors (MAE) and the IoU is 3.5 times bigger.



(a) IoU

(b) DTW

(c) Training Time

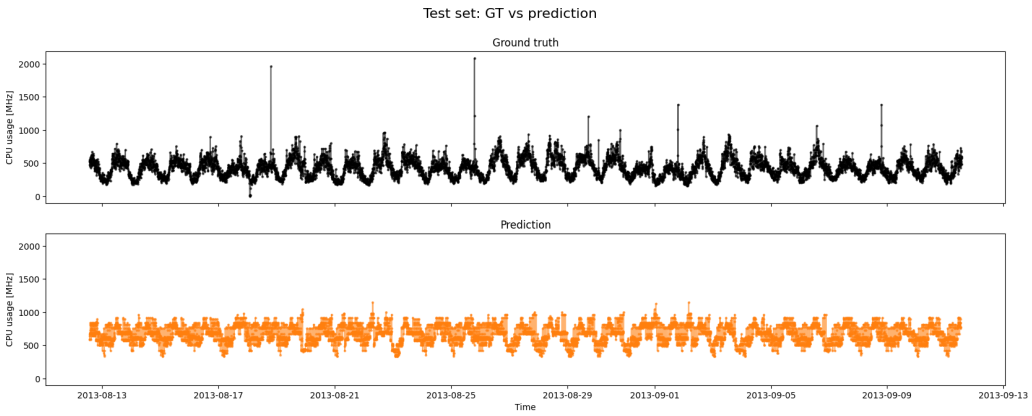
(d) Inference Time

(e) MAE, MAPE & RMSE

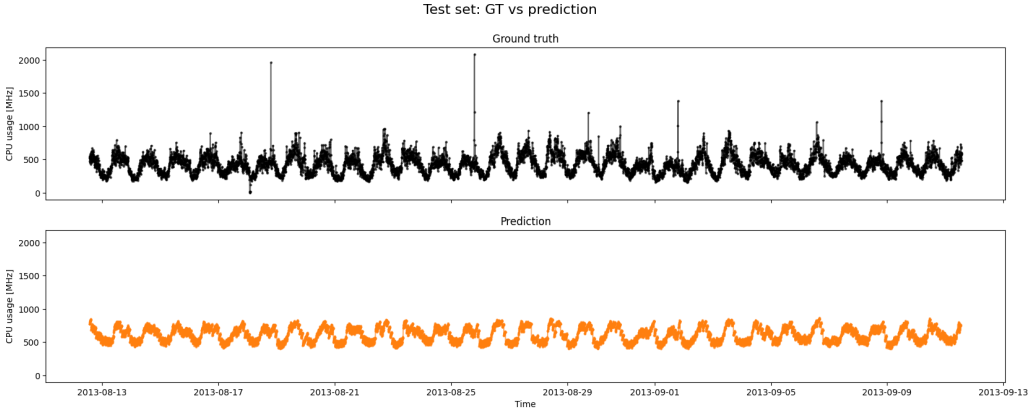
Figure 55. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the same cluster and similar pattern.

The LSTM model (see Fig. 60c) predicts the trend of the data with certain error. However the short-term pattern is not captured in the numeric approach.

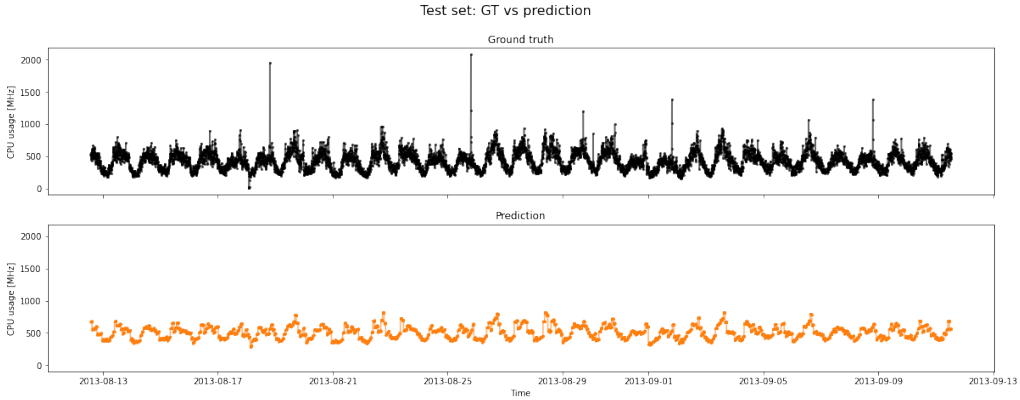
The autoencoder and the LRCN model cannot predict CPU utilization on a new unseen VM of a different shape. As could be observed in Fig. 60a and Fig. 60b, respectively, none



(a) Convolutional Autoencoder.



(b) Long Recurrent Convolutional Network (LRCN).

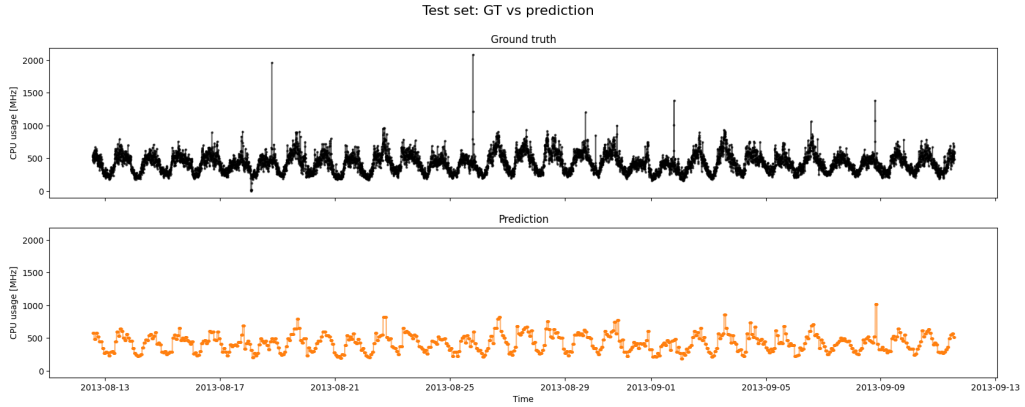


(c) LSTM.

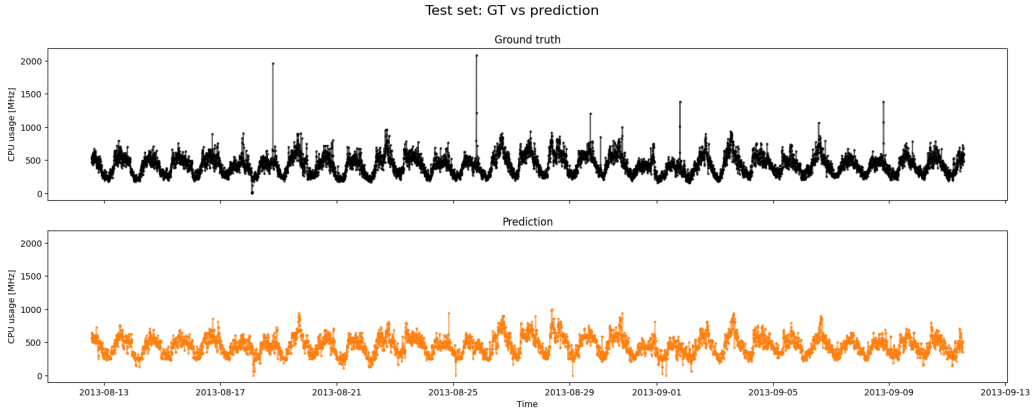
Figure 56. Comparison of real against the predicted by Convolutional Autoencoder, LRCN, and LSTM when inferring over the data of a virtual machine of the same cluster and similar pattern.

of the models is capable of accurately predicting the resource utilization of the new VM.

As a reference, the exponential smoothing model is presented (see Fig. 61a). The model is able to predict the trend when we see a long-term pattern (the data are constant for a certain number of consecutive timestamps) but cannot forecast the short-term pattern or the variance of the data.



(a) Exponential Smoothing.



(b) Video-frame (ConvLSTM).

Figure 57. Comparison of real against the predicted CPU utilization by Exponential Smoothing, and video-frame (ConvLSTM) when inferring over the data of a virtual machine of the same cluster and similar pattern.

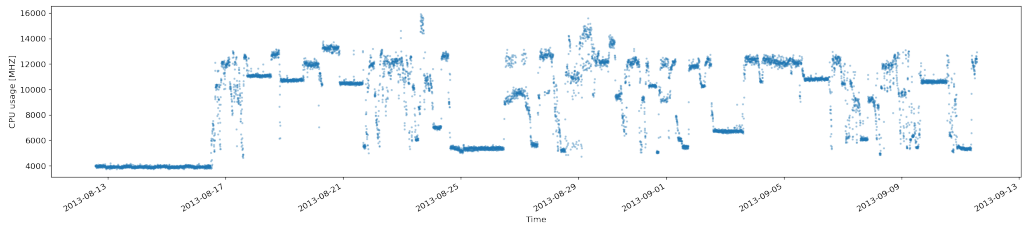
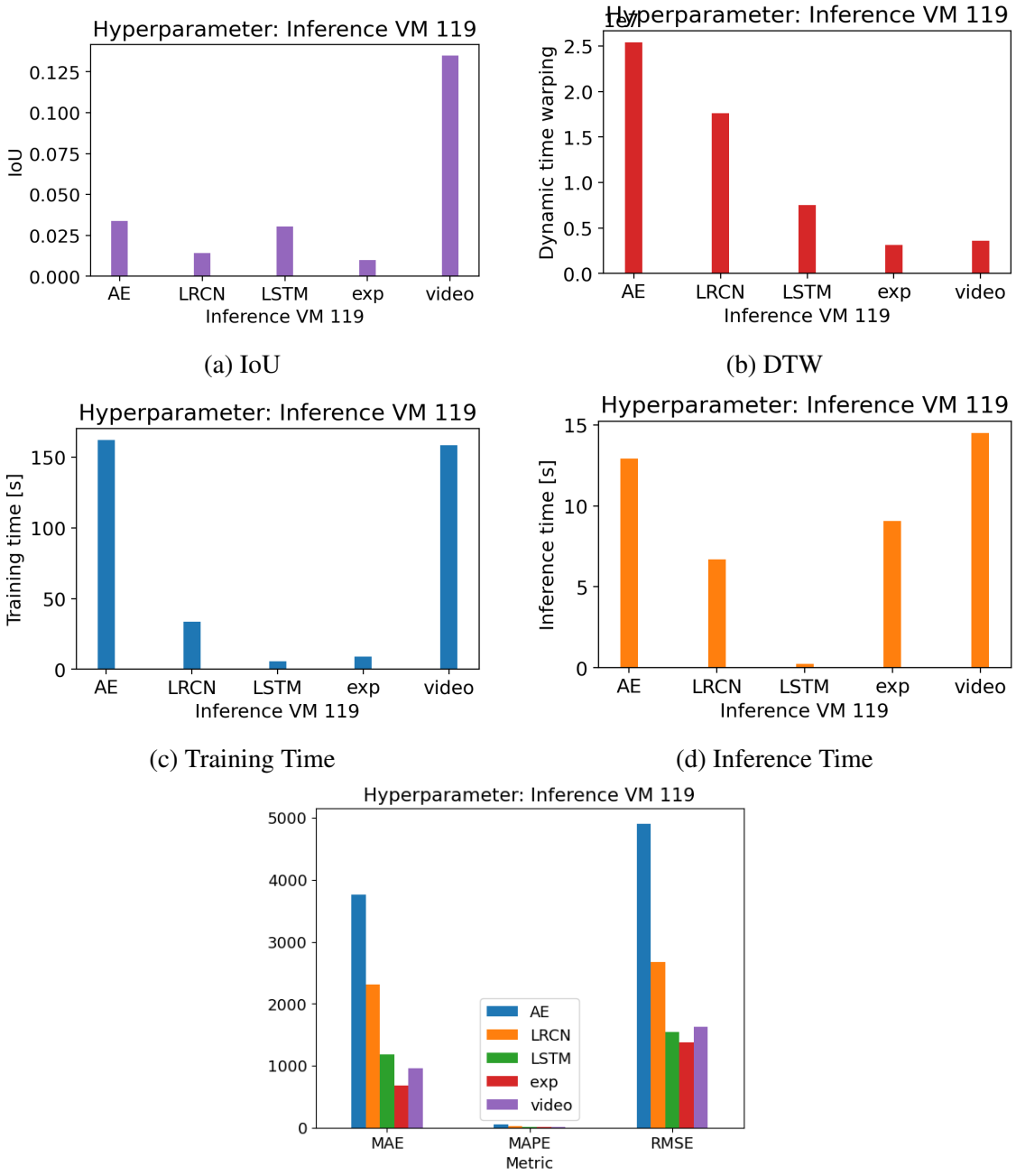


Figure 58. CPU utilization of Virtual Machine 119.

8.2.3 Different Cluster, Different Pattern

Table 20 and Fig. 63 show the metrics for the different image-driven models when inferred in VM 322 (see Fig. 62). The VM 322 does not have a “sinusoidal” shape, like the training VM and belongs to another cluster according to the one created in Chapter 5, thus it is considered to have a different shape and different from a different cluster than the training VM (VM 917).

The results are similar to those of the previous section. The video-frame prediction model



(a) IoU

(b) DTW

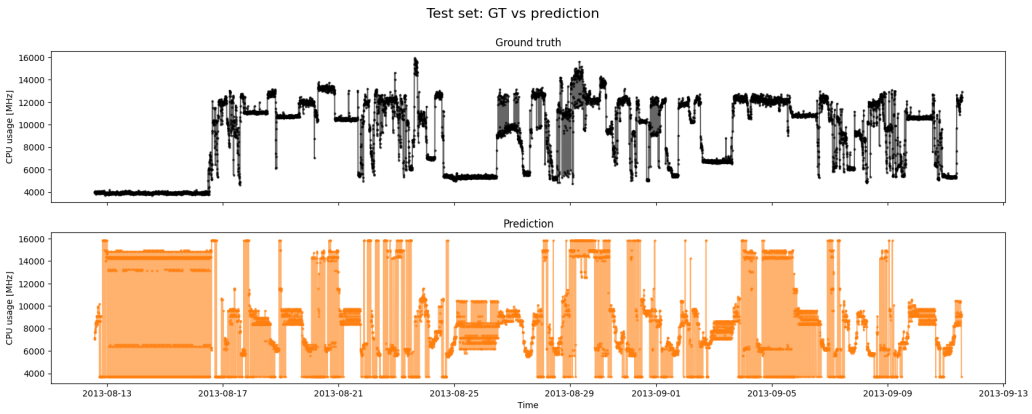
(c) Training Time

(d) Inference Time

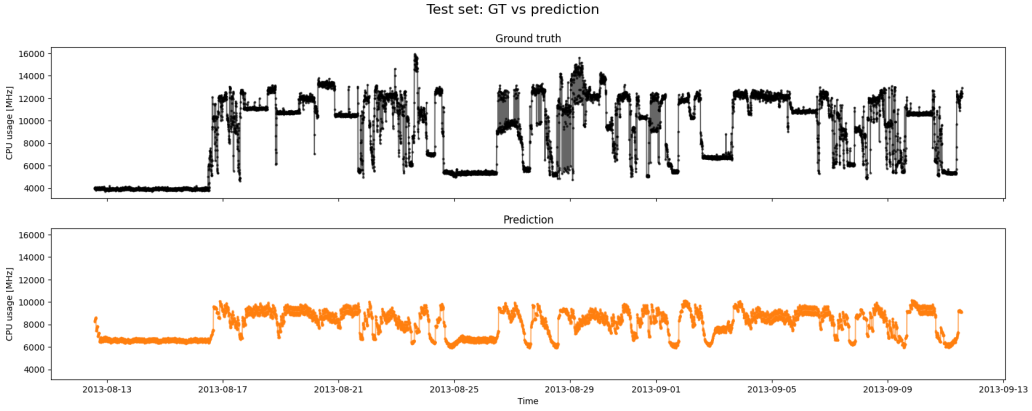
(e) MAE, MAPE & RMSE

Figure 59. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the same cluster and different pattern.

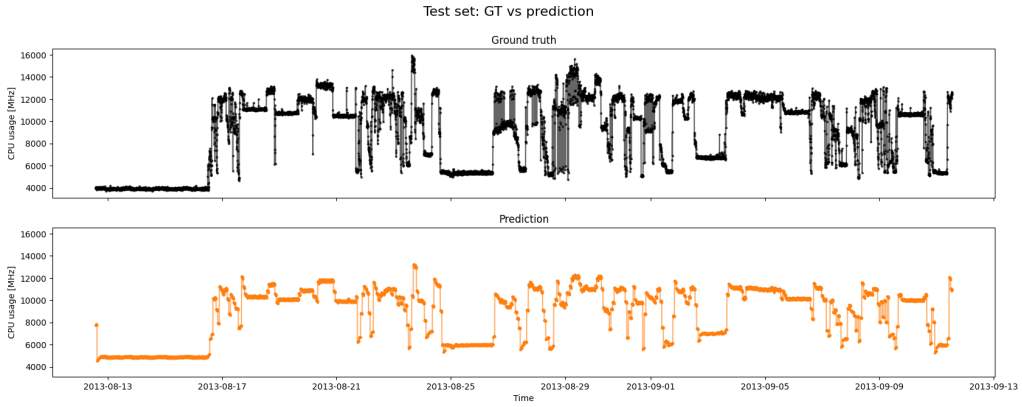
outperforms the other models according to numeric metrics (see Fig. 63e), IoU (see Fig. 63a) and DTW (see Fig. 63b). Fig. 65b shows how this model predicts the trend of the data and also the short-term pattern. The video-frame model outperforms the LSTM approach by 30% according to numeric errors (MAE) and has a IoU four times greater.



(a) Convolutional Autoencoder.



(b) Long Recurrent Convolutional Network (LRCN).

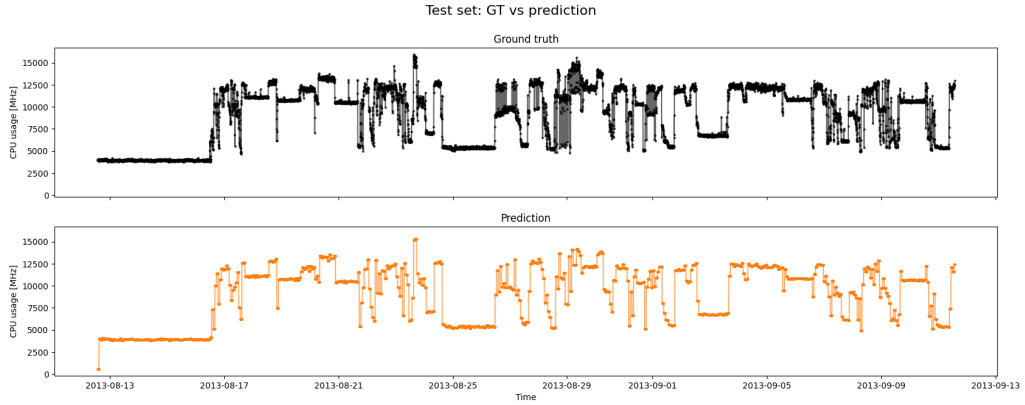


(c) LSTM.

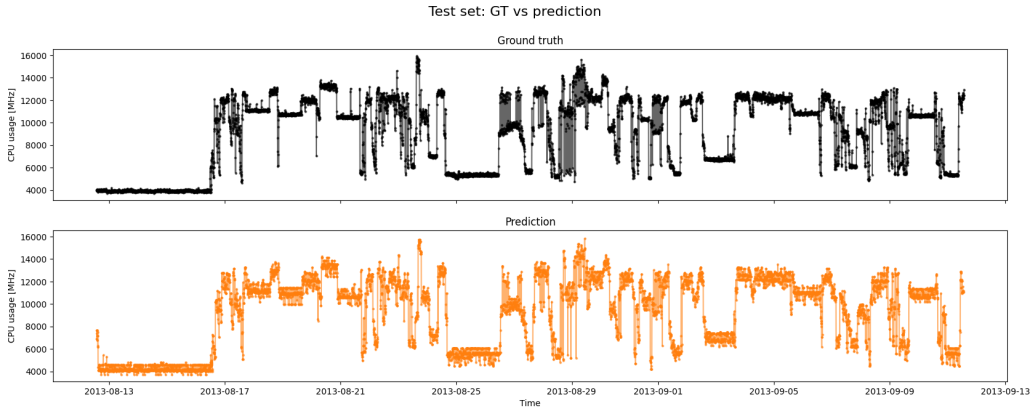
Figure 60. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, LRCN, and LSTM when inferring over the data of a virtual machine of the same cluster and different pattern.

Again, the LSTM model (see Fig. 64c) predicts the trend of the data with a certain error, especially when the utilization reaches maximum values, the model cannot predict those. The short-term pattern and the variance of the data are not captured.

The autoencoder predicts patterns that do not match the ground truth (observe Fig. 64a). The prediction of the LRCN model is very inaccurate as shown in Fig. 64b.



(a) Exponential Smoothing.



(b) Video-frame (ConvLSTM).

Figure 61. Comparison of real against the predicted CPU utilization by Exponential Smoothing, and video-frame (ConvLSTM) when inferring over the data of a virtual machine of the same cluster and different pattern.

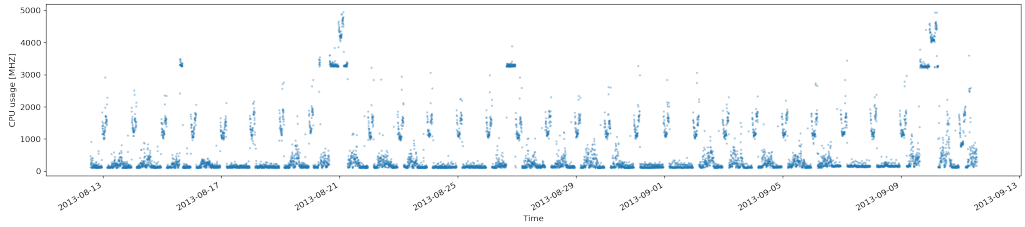
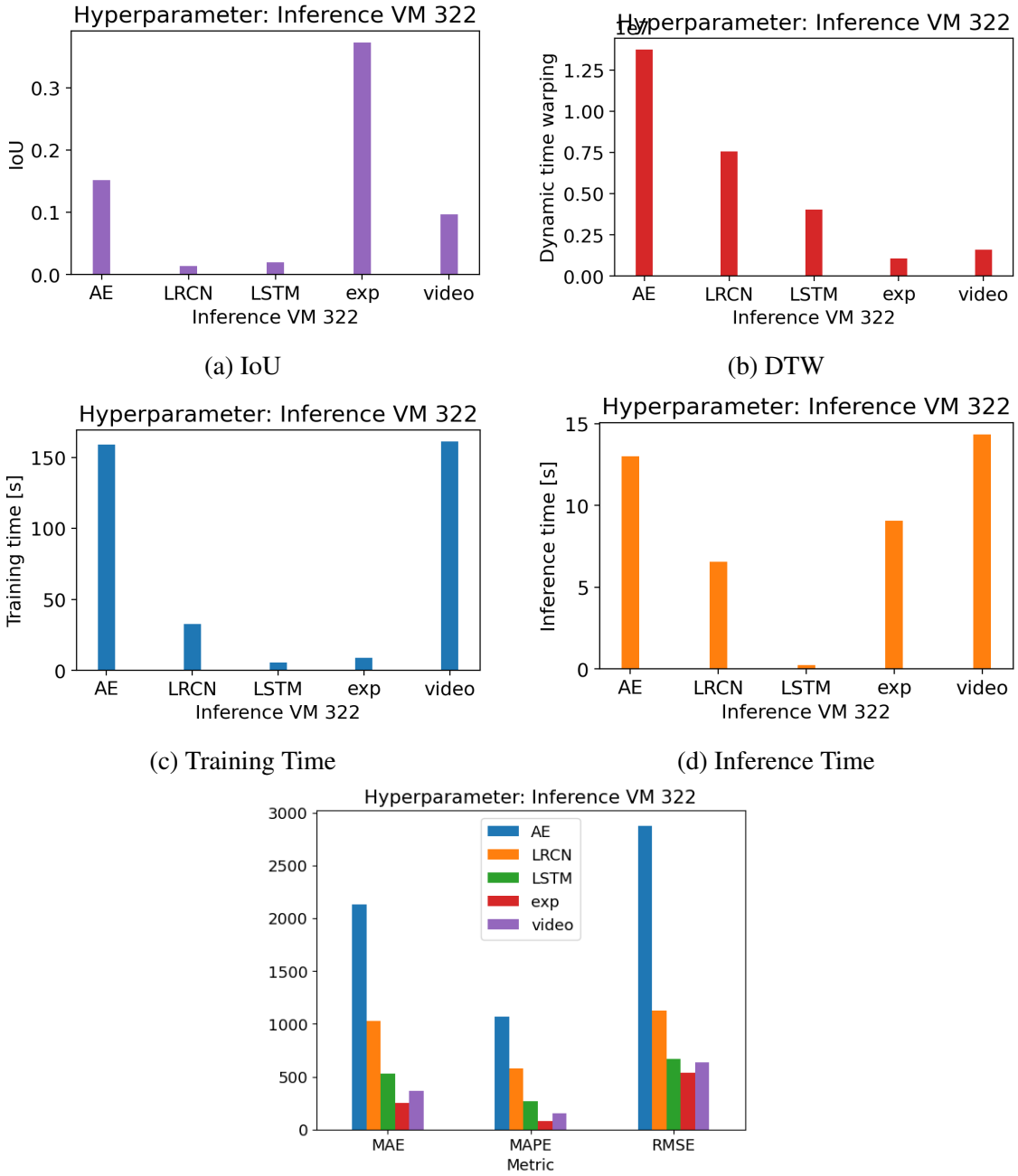


Figure 62. CPU utilization of Virtual Machine 322.

Table 20. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the different cluster and different pattern.

model	MAE	MAPE	RMSE	MASE	IoU	DTW	train time [s]	inference time [s]	model size [MB]
AE	2130.039	1071.620	2879.101	11.388	**0.152**	13723799.744	159.119	13.007	356.573
LRCN	1029.527	580.066	1127.313	5.504	0.014	7565198.862	32.656	6.553	1.052
LSTM	527.245	266.981	664.991	2.819	0.020	4036499.720	**5.784**	**0.256**	**0.023**
video-frame	**366.992**	**156.854**	**631.973**	1.962	0.097	**1605613.237**	161.017	14.327	34.692
exp smooth	251.936	79.386	538.089	4.262	0.372	1066755.491	9.059	9.059	nan

Again, we observe that when the shape of the new test set is not similar to the one of the training set, only the video-frame prediction model is capable of predicting both the short-term pattern and the trend of the data. Furthermore, we see that the clustering performed by the k-means algorithm is not convenient for finding VMs of similar patterns and more advanced pattern recognition techniques are needed in order to cluster the VMs of the Datacenter.



(a) IoU

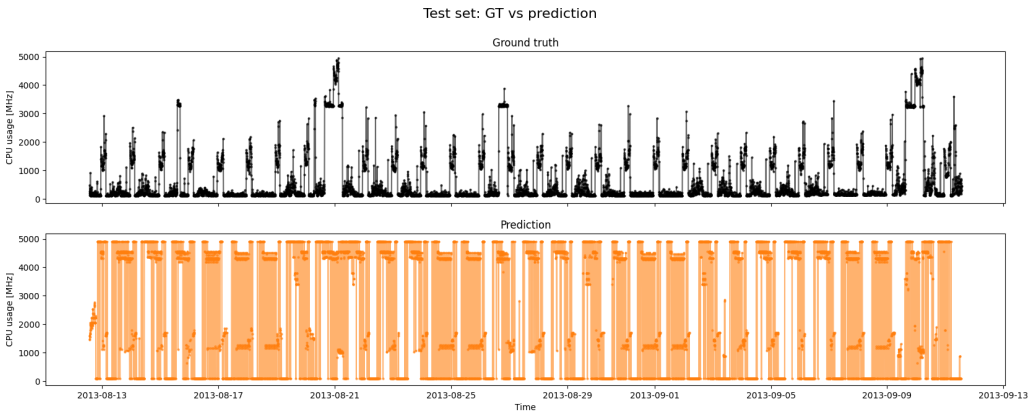
(b) DTW

(c) Training Time

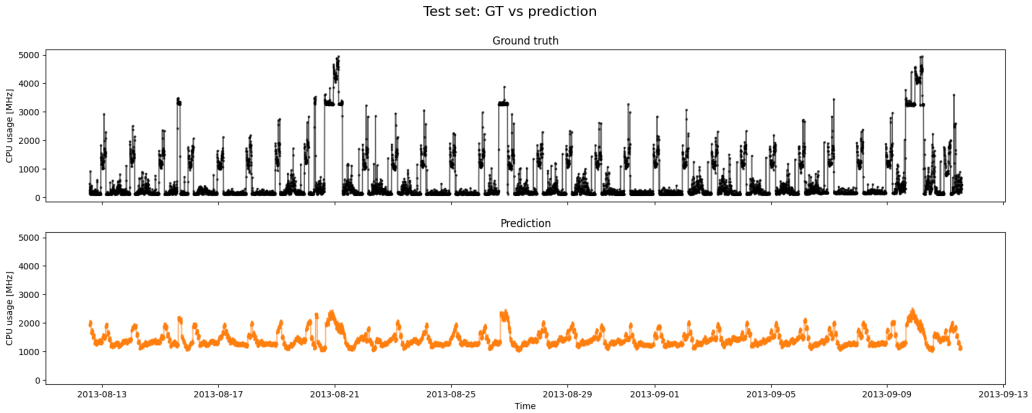
(d) Inference Time

(e) MAE, MAPE & RMSE

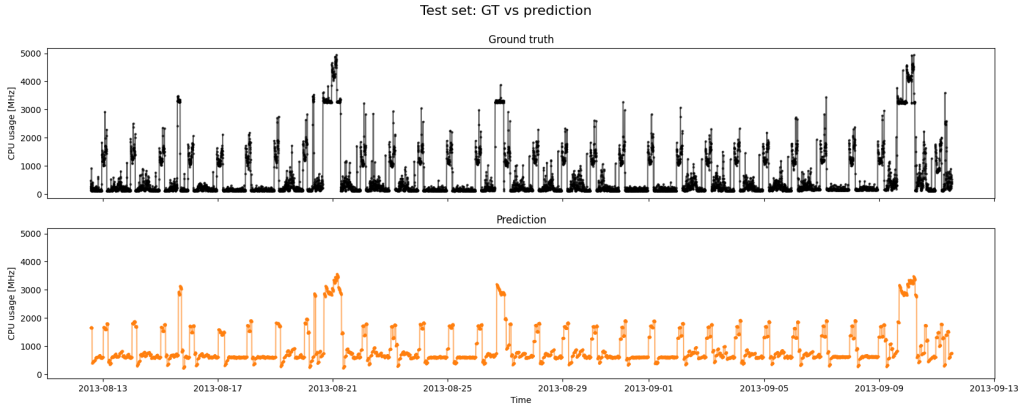
Figure 63. Comparison of prediction errors and performance metrics for the Convolutional Autoencoder, LRCN, LSTM, Exponential Smoothing and video-frame (ConvLSTM) models when performing inference over the data of a virtual machine of the different cluster and different pattern.



(a) Convolutional Autoencoder.



(b) Long Recurrent Convolutional Network (LRCN).

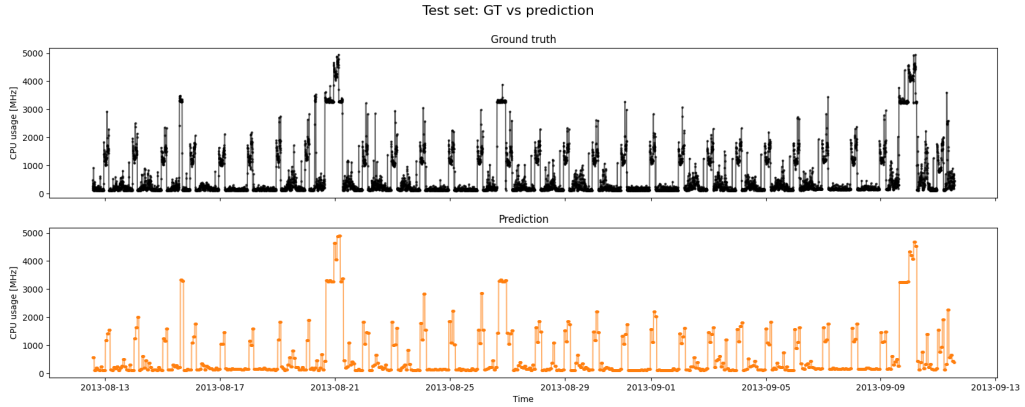


(c) LSTM.

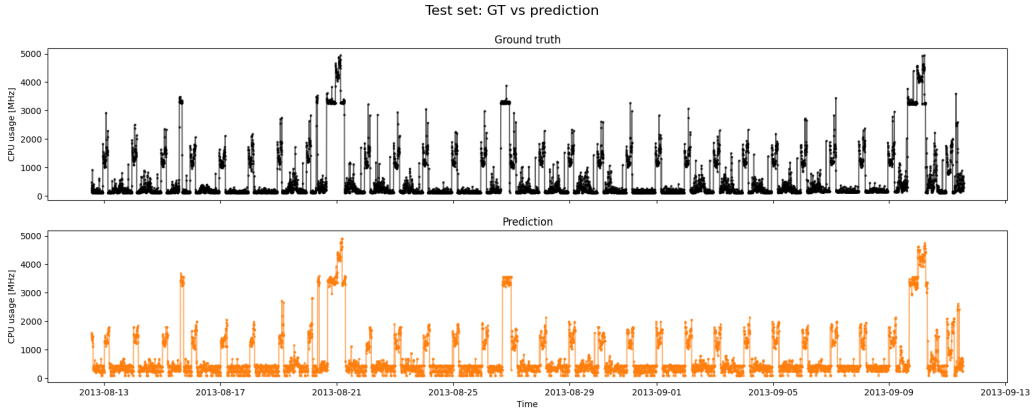
Figure 64. Comparison of real against the predicted CPU utilization by Convolutional Autoencoder, LRCN, and LSTM when inferring over the data of a virtual machine of the different cluster and different pattern.

8.3 Chapter Summary

In this chapter, we **compare** the different image-driven models with several baselines. The **image-driven models** are compared to a **numeric machine learning** approach such as **LSTM** and **traditional time-series forecasting models** such as **ARIMA** or **exponential smoothing**.



(a) Exponential Smoothing.



(b) Video-frame (ConvLSTM).

Figure 65. Comparison of real against the predicted CPU utilization by Exponential Smoothing, and video-frame (ConvLSTM) when inferring over the data of a virtual machine of the different cluster and different pattern.

First, the models are compared for different **forecasting horizons**. It is a very different problem to forecast tomorrow's weather than the next week's weather. A certain model might perform better for short-time forecasting periods and another one for longer forecasting periods. Usually, when the **forecasting horizon is very short (i.e., fh = 1)**, the model is simple and simpler models work best. This is observed during the experiments performed in this thesis. Simpler models such ARIMA or exponential smoothing work as well as a more complex model, and outperform a complex model in certain scenarios. However, it must be noted that while machine learning-based methods train for a longer time in a preproduction step, inference is generally fast in the implemented model. On the other hand, traditional methods such as ARIMA may train faster, but it is necessary to update the model in each iteration based on the past data during the inference stage. This could be observed during these experiments; ARIMA models usually accurately predict resource utilization, but updating the parameters, especially the moving average part, is very time-consuming and could be a significant disadvantage in certain scenarios, as may not be possible to implement due to time constraints.

When increasing the **forecasting horizon (i.e., fh = 8)**, we see how the problem is more complex and the results are slightly different. The traditional baseline (i.e., ARIMA and exponential smoothing) is capable of predicting the trend of the data, but these models do not capture the short-term patterns or the variance of the data. The video-frame prediction model best captures the short-term patterns and the variance of the data. For this forecasting horizon, the LSTM model predicts very accurately the trend of the data that yields good performance according to most metrics.

When the **forecasting horizon is even longer (i.e., fh = 16)**, the complexity of the problem reaches a point where the complexity of machine learning methods starts to be necessary. In this scenario, most algorithms perform similarly. The ARIMA, LRCN and LSTM models are able to predict the trend of the data accurately. However, only the autoencoder and the video-frame prediction models are capable of predicting the short-term pattern and the variance of the data.

For a **very long forecasting horizon (i.e., fh = 32)**, the complexity of the problem is high and none of the model performs properly. The autoencoder and the LRCN completely fail to forecast the trend of the data. The ARIMA or exponential smoothing models also make a poor forecast of the data trend. The video-frame predicts a similar trend to the ground truth, but it is not accurate.

It also analyzed in this chapter, the performance of the different models when the **training and test subset belong to different VMs**. A certain model may be very useful, whether it is trained over one VM and can correctly infer on a different VM. We have explored how the different models work when the VM have similar **patterns** and when they belong to the same **cluster**. Only machine learning models save "information" about the training data and use it for inference. The exponential smoothing model is just a weighted average of previous timestamps and the ARIMA model is updating its parameters in each iteration, based on previous data.

When both VMs belong to the **same cluster** and also have a **similar shape**, the video-frame prediction model is able to predict the trend and short-term patterns of the data. The LRCN and the LSTM forecast the trend of the data with some error. The autoencoder does not properly work in this scenario.

We have also shown that when the **pattern of the new VM is different**, it does not matter whether it belongs to the same cluster generated by k-means or not. The video-frame prediction model surprisingly predicts both the short-term patterns and the trend of the data accurately. Only the variance of the data is slightly overestimated. The LSTM predicts the

trend of the data with certain offset and error. The autoencoder and the LRCN performance is unsatisfactory. We could see how the LSTM cell helps to save the sequential information of the previous data to make a more accurate forecast. We have shown that whether two VM have similar patterns it is possible to infer in the second one just with the model trained in the first one. However, we have also observed how the clusters generated by k-means do not classify the VM according to its pattern similarity. Thus, more advanced pattern recognition techniques are needed to correctly cluster the different VM in clusters with a similar pattern. Whether this goal is achieved, it would be possible to have a model per cluster and infer in all the other similar VMs with high accuracy using the same model.

Throughout this chapter, we have observed how certain models best predict the **short-term pattern and the variance of the data**, and others best forecast **the trend of the data**. Therefore, instead of selecting only one model, exploring ensemble methods could yield the best performance, leveraging the best of each world.

9. Conclusion

9.1 Summary

This thesis presented a novel approach to forecasting future cloud resource utilization leveraging current state-of-the-art computer vision and machine learning methods. Datacenter operators and users of cloud environments, such as virtual machines, need mechanisms to decide how many resources to utilize for workload execution. Thus, having the ability to accurately predict the demand for hardware resources in the future is very important to achieve efficient resource management and cost efficiency in cloud environments. However, the way workloads utilize hardware resources at the cloud is very complex and often times random. This non-uniformity and complexity of cloud resource usage, lowers the predictive capabilities of traditional time series forecasting methods. In response, recent solutions rely on more sophisticated machine learning algorithms, which introduce a new challenge of providing accurate predictions in exchange for short training and inference times.

Recent research in the financial domain demonstrates how using an image representation of time series data and related image-based methodologies can lead to more reliable and effective forecasting. For that purpose, this thesis has investigated the use of images, computer vision, and machine learning approaches to forecast future resource usage in cloud environments. We proposed an image-based prediction pipeline that visualizes data in a sophisticated fashion, predicts resource usage using image-based machine learning approaches similar to those used in video frame prediction, and then decomposes the predicted images back to numeric predictions. We evaluated various image-based machine learning methods and compared them against traditional and other machine learning-based forecasting methods, using a real dataset of cloud resource consumption over the span of two months. Our analysis shows that methods used for video frame prediction can accurately forecast resource utilization for extended periods of time in the future and learn both the short term patterns and overall trend of the data. In addition, these video frame prediction methods can reliably predict resource consumption even when the training and inference datasets exhibit completely different patterns, something that other image-based, non-image-based, and traditional forecasting methods cannot currently do.

In conclusion, this thesis identifies and demonstrates how the use of an image-based pipeline for predicting cloud resource consumption outperforms current approaches, particularly for very challenging cases such as long forecasting horizons and inference over unseen data with different trends from the training dataset. This thesis aims to lay the foundations for future use of computer vision, visualization and image-based pipelines inside systems software.

9.2 Lessons Learned

Next, we summarize valuable lessons learned that derive across the various aspects that this thesis explores.

Datacenter resource utilization is extremely variable, rendering unsupervised clustering methods ineffective.

The analysis presented in this thesis uses a dataset that captures real traces of datacenter resource utilization [40], instead of synthetic data. The dataset captures the resource utilization across thousands of virtual machines every 5 minutes between August and September 2013. Since the dataset is massive, we rely on unsupervised machine learning clustering methods, such as k-means, to extract groups of virtual machines that share similar trends in resource utilization across time. Our analysis, presented in Chapter 5, shows that k-means fails to create clusters that capture distinct patterns. Therefore, we reside in our own abilities and manually group together virtual machines with similar patterns, such as ones resembling a sinusoidal distribution, as depicted in Section 5.3.3. In conclusion, the exploration of a massive real dataset is challenging and it may require some manual exploration to extract insights, since unsupervised clustering methods can be ineffective.

Effective image composition should have minimal information loss and be scaled to capture any repeating pattern.

The experiments carried out throughout this thesis have shown that an effective image representation of the numeric data is essential for the image-based machine learning pipeline to make accurate predictions. Our method presented in Chapter 6 visualizes data in a sophisticated way, which enables image-based machine learning methods to predict resource consumption, and then decomposes the predicted images back to numeric predictions to use as the final forecast. Our exploration of various image sizes and pixel-to-data mapping shows that it is very important to have a sophisticated visualization. In particular, ensuring that the images clearly visualize any repeating pattern and there is

minimal information loss is what enables the image-based machine learning models to provide accurate predictions. In other words, for the machine learning methods to learn a pattern from an image, the image should clearly capture and visualize that pattern.

Video frame prediction models are able to predict data over long forecasting horizons in the future.

During the thorough comparison performed in Chapter 8, we have observed how the length of the forecasting horizon determines the complexity of a forecasting problem. When the forecasting horizon is short, the forecasting problem is rather simple, and classical methods perform as well as a more complex model or even better. However, we observed that when the forecasting horizon is longer, the complexity of the problem increases and we need more sophisticated models. In these scenarios, image-based models clearly outperform traditional methods and numeric machine learning approaches. Furthermore, when the forecasting horizon is even longer, the video frame prediction model is the only one capable of properly predicting the trend and shape of the data.

Traditional metrics that capture the prediction error may not effectively capture the learning capabilities of a model.

The analyses performed throughout this thesis in Chapters 7 and 8 shows how traditional numeric metrics that capture the prediction error, such as the mean absolute error, may be misleading in terms of determining which model effectively learns the provided data trends. For example, traditional forecasting methods may end up predicting values close to the mean of the data, which results in a low mean absolute error. For this reason, we visually inspected the predicted shape of the data and we observed that the image-based models best capture the shape of the data and the short-term pattern. Thus, we ended up using error metrics that capture image-based differences, such as the Intersection over Union (IoU) or ones that are specific to time series comparison, such as the Dynamic Time Warping (DTW). In conclusion, we highly encourage the use of such metrics and the visual inspection of the predicted data to validate the learning capabilities of a forecasting model.

Video frame prediction models are able to capture both the short term patterns and the overall trend of the data.

Using various evaluation metrics and comparing across different image-based, non-image-based machine learning and traditional forecasting methods, we conclude that the methods used for video frame prediction are the ones that learn and accurately predict both short-term patterns and the overall trend of the data. This highlights the benefit of using purely

image-based pipelines for predicting cloud resource efficiency.

Video frame prediction models are able to predict unseen data, that even exhibit completely different patterns compared to the training dataset.

Developing a model for each cloud user or virtual machine of a datacenter could be very time-consuming. Thus, using trained models in new unseen data or data from a virtual machine that even exhibits completely different patterns compared to the training dataset may be very convenient for reducing cost and optimizing resources. The experiments carried out in Chapter 8 show that the proposed image-based machine learning method, typically used in video frame prediction, can accurately forecast resource utilization, even when the training and inference datasets exhibit completely different characteristics, something that is currently not possible using other image-based, non-image-based and traditional forecasting methods.

9.3 Discussion and Future Directions

Next, we discuss future directions of the thesis, based on the analysis presented, the explored aspects of the problem and the methods taken into consideration.

Exploring other real datasets and features.

We have shown that an image-driven approach can effectively forecast future cloud resource utilization. In this thesis, we utilize real data of resource consumption of various virtual machines that are deployed in the same datacenter, captured in the widely used Bitbrains dataset [40]. In future work, it is worth validating these models and algorithms in new datasets such as Google Cluster Workload Traces [48] and Alibaba Cluster Trace Program [49], which can potentially reveal new patterns and data behaviors of cloud resource utilization.

During this thesis, the different resource utilization within the datacenter (e.g., CPU, memory, disk) do not have relevant correlation between them, and thus, the different features are analyzed independently. The time series are univariate. However, future work could explore the representation of multiple features in an image. For example, using different channels instead of grayscale images, and the intensity of a channel as an extra feature, instead of using binary images.

Exploring other clustering methods for dataset exploration.

Throughout this thesis, we have shown that unsupervised machine learning clustering methods, such as k-means, fail to create clusters of virtual machines that capture distinct patterns for the given dataset. Therefore, we reside in our own abilities and manually group together virtual machines with similar patterns. Future work could explore other datasets or the use of more sophisticated pattern recognition algorithms to create clusters of virtual machines that have a similar shape or trend.

Exploring other image representations.

In this thesis, we have shown that an effective image representation of the raw data is essential for the image-based machine learning pipeline to make accurate predictions. Our approach visualizes the time series data of cloud resource consuption 'as-is', in two dimensional images where the x-axis corresponds to time and the y-axis to the value of resource usage at the specific time. However, in the time-series domain, there are other techniques to represent time-series data as images, such as Gramian Angular Field (GAF) and recurrence plots. Decomposing such images and creating the reserve mapping of these representations back to raw time series values is not trivial. Future work could explore ways to decompose such time series representations and analyze whether they contribute to higher prediction accuracy when integrated in the image-based forecasting pipeline.

Exploring the use of other models and transfer learning techniques.

Future work could explore whether increasing the depth of the machine learning models and employing other state-of-the-art computer vision models might increase the accuracy of the predictions. Throughout the thesis we have observed that more complex models such as the video-frame prediction are able to make forecasts in unseen data that have completely different patterns from the training dataset. Future work may explore deeper models, that usually need bigger datasets, trained over several virtual machines instead of one. Furthermore, although these images are different from common object datasets, such as COCO [64], it is worth exploring whether transfer learning techniques can be effective, where a model that is pretrained on one dataset is used to perform inference on another dataset [65].

Exploring the use of ensemble machine learning methods.

Our analysis shows that some models perform best in predicting the short-term pattern of the data and others in forecasting the trend of the data, while the video frame prediction methods do so for both. Future work could explore the use of ensemble methods [66] to combine different machine learning methods, neural network architectures and build a

single forecasting model that is robust across various datasets and deployment scenarios.

Bibliography

[1] Gurleen Kaur, Anju Bala, and Inderveer Chana. "An intelligent regressive ensemble approach for predicting resource usage in cloud computing". In: *Journal of Parallel and Distributed Computing* 123 (Jan. 2019), pp. 1–12. ISSN: 07437315. DOI: 10.1016/j.jpdc.2018.08.008.

[2] Arfa Muteeh, Muhammad Sardaraz, and Muhammad Tahir. "MrLBA: multi-resource load balancing algorithm for cloud computing using ant colony optimization". In: *Cluster Computing* 24 (4 Dec. 2021), pp. 3135–3145. ISSN: 1386-7857. DOI: 10.1007/s10586-021-03322-3.

[3] Sukhpal Singh and Inderveer Chana. "QoS-Aware Autonomic Resource Management in Cloud Computing". In: *ACM Computing Surveys* 48 (3 Feb. 2016), pp. 1–46. ISSN: 0360-0300. DOI: 10.1145/2843889.

[4] Sean J Taylor and Benjamin Letham. "Forecasting at scale". In: (2017). DOI: 10.7287/peerj.preprints.3190v1.

[5] Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. "Predicting cloud resource utilization". In: Association for Computing Machinery, Inc, Dec. 2016, pp. 37–42. ISBN: 9781450346160. DOI: 10.1145/2996890.2996907.

[6] Soukaina Ouhame, Youssef Hadi, and Arif Ullah. "An efficient forecasting approach for resource utilization in cloud data center using CNN-LSTM model". In: *Neural Computing and Applications* 33 (16 Aug. 2021), pp. 10043–10055. ISSN: 14333058. DOI: 10.1007/s00521-021-05770-9.

[7] Shaifu Gupta, A. D. Dileep, and Timothy A. Gonsalves. "Online Sparse BLSTM Models for Resource Usage Prediction in Cloud Datacentres". In: *IEEE Transactions on Network and Service Management* 17 (4 Dec. 2020), pp. 2335–2349. ISSN: 19324537. DOI: 10.1109/TNSM.2020.3013922.

[8] Mahendra Pratap Yadav, Rohit, and Dharmendra Kumar Yadav. "Resource Provisioning Through Machine Learning in Cloud Services". In: *Arabian Journal for Science and Engineering* 47 (2 Feb. 2022), pp. 1483–1505. ISSN: 2193-567X. DOI: 10.1007/s13369-021-05864-5.

[9] Akindele A. Bankole and Samuel A. Ajila. “Predicting cloud resource provisioning using machine learning techniques”. In: *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. 2013, pp. 1–4. DOI: 10.1109/CCECE.2013.6567848.

[10] T Hastie et al. *Elements of Statistical Learning, The*.

[11] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and practice*. OTexts, 2021.

[12] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. “The M4 competition: 100,000 time series and 61 forecasting methods”. In: *International Journal of Forecasting* 36.1 (2020), pp. 54–74. DOI: 10.1016/j.ijforecast.2019.04.014.

[13] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. “An Experimental Review on Deep Learning Architectures for Time Series Forecasting”. In: *International Journal of Neural Systems* 31 (3 Mar. 2021). ISSN: 17936462. DOI: 10.1142/S0129065721300011.

[14] David Salinas, Valentin Flunkert, and Jan Gasthaus. “DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks”. In: (Apr. 2017). URL: http://arxiv.org/abs/1704.04110.

[15] Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. “Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach”. In: (Oct. 2017). URL: http://arxiv.org/abs/1710.03222.

[16] Naftali J Cohen P Morgan AI Research New York et al. *Visual Time Series Forecasting: An Image-driven Approach*. 2021.

[17] Srijan Sood et al. “Visual Time Series Forecasting: An Image-driven Approach”. In: (Nov. 2020). DOI: 10.1145/3490354.3494387. URL: http://arxiv.org/abs/2011.09052%20http://dx.doi.org/10.1145/3490354.3494387.

[18] Anupama K C, Shivakumar B, and Nagaraja R. “Resource Utilization Prediction in Cloud Computing using Hybrid Model”. In: *International Journal of Advanced Computer Science and Applications* 12 (4 2021). ISSN: 21565570. DOI: 10.14569/IJACSA.2021.0120447.

[19] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[20] *Understanding LSTM networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[21] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. “Multi-column Deep Neural Networks for Image Classification”. In: *CoRR* abs/1202.2745 (2012). arXiv: 1202.2745. URL: http://arxiv.org/abs/1202.2745.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[23] Mrgrhn. *Convolutional autoencoders (CAE) with tensorflow*. Jan. 2021. URL: https://ai.plainenglish.io/convolutional-autoencoders-cae-with-tensorflow-97e8d8859cbe.

[24] Xingjian Shi et al. “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: (June 2015). URL: http://arxiv.org/abs/1506.04214.

[25] Jeff Donahue et al. “Long-term Recurrent Convolutional Networks for Visual Recognition and Description”. In: (Nov. 2014). URL: http://arxiv.org/abs/1411.4389.

[26] S. Sarikaa, S. Niranjana, and K. Sri Vishnu Deepika. “Time Series Forecasting of Cloud Resource Usage”. In: Institute of Electrical and Electronics Engineers Inc., 2021, pp. 372–382. ISBN: 9781665414739. DOI: 10.1109/ICCCA52192.2021.9666444.

[27] Oscar Reyes and Sebastián Ventura. “Performing Multi-Target Regression via a Parameter Sharing-Based Deep Network”. In: *International Journal of Neural Systems* 29 (09 Nov. 2019), p. 1950014. ISSN: 0129-0657. DOI: 10.1142/S0129065719500140X.

[28] Alok Sharma et al. “DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture”. In: *Scientific Reports* 9 (1 Dec. 2019). ISSN: 20452322. DOI: 10.1038/s41598-019-47765-6.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional Neural Networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[30] Naftali Cohen, Tucker Balch, and Manuela Veloso. “Trading via image classification”. In: Association for Computing Machinery, Inc, Oct. 2020. ISBN: 9781450375849. DOI: 10.1145/3383455.3422544.

[31] Silvio Barra et al. “Deep learning and time series-To-image encoding for financial forecasting”. In: *IEEE/CAA Journal of Automatica Sinica* 7 (3 May 2020), pp. 683–692. ISSN: 23299274. DOI: 10.1109/JAS.2020.1003132.

[32] Xixi Li, Yanfei Kang, and Feng Li. “Forecasting with time series imaging”. In: *Expert Systems with Applications* 160 (Dec. 2020). ISSN: 09574174. DOI: 10.1016/j.eswa.2020.113680.

[33] Zhen Zeng, Tucker Balch, and Manuela Veloso. “Deep Video Prediction for Time Series Forecasting”. In: (Feb. 2021). URL: http://arxiv.org/abs/2102.12061.

[34] Meinard Müller. *Information retrieval for music and motion*. Springer, 2007.

[35] Charu C Aggarwal. *Data mining*. Springer, 2015.

[36] URL: https://commons.wikimedia.org/wiki/File:Euclidean_vs_DTW.jpg.

[37] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) challenge”. In: *International Journal of Computer Vision* 88.2 (2009), pp. 303–338. DOI: 10.1007/s11263-009-0275-4.

[38] Hamid Rezatofighi et al. “Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 658–666. DOI: 10.1109/CVPR.2019.00075.

[39] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[40] 2022. URL: http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains.

[41] Jyoti Shetty and G Shobha. “An ensemble of automatic algorithms for forecasting resource utilization in cloud”. In: *2016 Future Technologies Conference (FTC)*. 2016, pp. 301–306. DOI: 10.1109/FTC.2016.7821626.

[42] Vincent van Beek et al. “Self-Expressive Management of Business-Critical Workloads in Virtualized Datacenters”. In: *Computer* 48.7 (2015), pp. 46–54. DOI: 10.1109/MC.2015.206.

[43] Yaqiu Liu et al. “Enhancing Energy-Efficient and QoS Dynamic Virtual Machine Consolidation Method in Cloud Environment”. In: *IEEE Access* 6 (2018), pp. 31224–31235. DOI: 10.1109/ACCESS.2018.2835670.

[44] Ali Pahlevan et al. “Integrating Heuristic and Machine-Learning Methods for Efficient Virtual Machine Allocation in Data Centers”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.8 (2018), pp. 1667–1680. DOI: 10.1109/TCAD.2017.2760517.

[45] Jalal Khamse-Ashari et al. “An Efficient and Fair Multi-Resource Allocation Mechanism for Heterogeneous Servers”. In: *IEEE Transactions on Parallel and Distributed Systems* 29.12 (2018), pp. 2686–2699. DOI: 10.1109/TPDS.2018.2841915.

[46] Zoltán Ádám Mann. “Multicore-Aware Virtual Machine Placement in Cloud Data Centers”. In: *IEEE Transactions on Computers* 65.11 (2016), pp. 3357–3369. DOI: 10.1109/TC.2016.2529629.

[47] Seyedhamid Mashhadi Moghaddam et al. “Energy-Efficient and SLA-Aware Virtual Machine Selection Algorithm for Dynamic Resource Allocation in Cloud Data Centers”. In: *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. 2018, pp. 103–113. DOI: 10.1109/UCC.2018.00019.

[48] *Google Cluster Workload traces 2019*. URL: https://research.google/tools/datasets/google-cluster-workload-traces-2019/.

[49] Alibaba. *Alibaba/clusterdata: Cluster data collected from production clusters in Alibaba for Cluster Management Research*. URL: https://github.com/alibaba/clusterdata.

[50] Siqi Shen, Vincent Van Beek, and Alexandru Iosup. “Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters”. In: *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2015, pp. 465–474. DOI: 10.1109/CCGrid.2015.60.

[51] 2022. URL: https://en.wikipedia.org/wiki/Moving_average.

[52] S. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/tit.1982.1056489.

[53] Robert L. Thorndike. “Who belongs in the family?” In: *Psychometrika* 18.4 (1953), pp. 267–276. DOI: 10.1007/bf02289263.

[54] Li Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[55] Guillaume Alain and Yoshua Bengio. *What Regularized Auto-Encoders Learn from the Data Generating Distribution*. 2012. DOI: 10.48550/ARXIV.1211.4246. URL: https://arxiv.org/abs/1211.4246.

[56] Lovedeep Gondara. “Medical Image Denoising Using Convolutional Denoising Autoencoders”. In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, Dec. 2016. DOI: 10.1109/icdmw.2016.0041. URL: https://doi.org/10.1109%2Ficdmw.2016.0041.

[57] Yijun Li et al. *Generative Face Completion*. 2017. DOI: 10.48550/ARXIV.1704.05838. URL: https://arxiv.org/abs/1704.05838.

[58] Wei Bao, Jun Yue, and Yulei Rao. “A deep learning framework for financial time series using stacked autoencoders and long-short term memory”. In: *PLOS ONE* 12.7 (2017), e0180944. DOI: 10.1371/journal.pone.0180944.

[59] Pablo Romeu et al. “Stacked denoising auto-encoders for short-term time series forecasting”. In: *Springer Series in Bio-/Neuroinformatics* (2015), pp. 463–486. DOI: 10.1007/978-3-319-09903-3_23.

[60] André Gensler et al. “Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks”. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2016, pp. 002858–002865. DOI: 10.1109/SMC.2016.7844673.

[61] Keras Team. *Keras documentation: Next-Frame Video Prediction with Convolutional LSTMs*. 2022. URL: https://keras.io/examples/vision/conv_lstm/.

[62] Taylor G. Smith et al. *pmdarima: ARIMA estimators for Python*. [Online; accessed <today>]. 2017–. URL: http://www.alkaline-ml.com/pmdarima.

[63] Skipper Seabold and Josef Perktold. “statsmodels: Econometric and statistical modeling with python”. In: *9th Python in Science Conference*. 2010.

[64] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[65] Stevo Bozinovski. “Reminder of the First Paper on Transfer Learning in Neural Networks, 1976”. In: *Informatica* 44 (3 Sept. 2020). ISSN: 1854-3871. DOI: 10.31449/inf.v44i3.2828.

[66] D. Opitz and R. Maclin. “Popular Ensemble Methods: An Empirical Study”. In: *Journal of Artificial Intelligence Research* 11 (Aug. 1999), pp. 169–198. ISSN: 1076-9757. DOI: 10.1613/jair.614.