



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόβλεψη Προτύπων Πρόσβασης Μνήμης με την Χρήση
Μηχανικής Μάθησης και Ορασης Υπολογιστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αριστοτέλης-Γέωργιος Συμπέθερος

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2024



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόβλεψη Προτύπων Πρόσβασης Μνήμης με την Χρήση
Μηχανικής Μάθησης και Ορασης Υπολογιστών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αριστοτέλης-Γέωργιος Συμπέθερος

Επιβλέπων: Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26^η Μαρτίου 2024.

(Υπογραφή)

(Υπογραφή)

(Υπογραφή)

.....
Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

.....
Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

.....
Σωτήριος Ξύδης
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2024



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

**Learning Memory Access Patterns Using Machine Learning
and Computer Vision**

DIPLOMA THESIS

Aristotelis-Georgios Sympetheros

Supervisor: Panayiotis Tsanakas
Professor at NTUA

Athens, March 2024

(Υπογραφή)

.....
Αριστοτέλης-Γεώργιος Συμπέθερος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Αριστοτέλης-Γεώργιος Συμπέθερος, 2024.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Τα σύγχρονα υπολογιστικά συστήματα, από τις προσωπικές συσκευές έως τα ισχυρά κέντρα δεδομένων, έχουν σημειώσει αξιοσημείωτη πρόοδο στην επεξεργαστική τους ισχύ. Αυτή η σημαντική πρόοδος, ωστόσο, αυξάνει την ανισότητα μεταξύ της υπολογιστικής ισχύος και της ταχύτητας πρόσβασης στη μνήμη, οδηγώντας σε μείωση της δυνητικής απόδοσης. Σε επίπεδο επεξεργαστή, αυτό μεταφράζεται σε χαμένους υπολογιστικούς κύκλους. Σε επίπεδα συστήματος, η μεγαλύτερη υπολογιστική ισχύ οδηγεί σε αύξηση της απαιτούμενης μνήμης και συνεπώς ανάγκη για χρήση μη πτητικής μνήμης (Non-Volatile Memory, NVM), οδηγώντας σε Υβριδικές Λύσεις Μνήμης (Hybrid Memory Systems, HMS). Η αρχιτεκτονική υπολογιστών χρησιμοποιεί μεθόδους πρόβλεψης για τον μετριασμό των προαναφερθέντων προβλημάτων. Στο επίπεδο του επεξεργαστή, αυτό περιλαμβάνει την προφόρτωση δεδομένων (data prefetching), όπου τα δεδομένα που θα χρειαστούν σε επόμενο χρόνο φορτώνονται εκ των προτέρων στην κρυφή μνήμη. Σε επίπεδο συστήματος, χρησιμοποιούνται διάφορες τεχνικές που σχετίζονται με τις σελίδες μνήμης (χρονοπρογραμματισμός, μεταφορά, αντικατάσταση κ.λπ.) για τη βελτιστοποίηση της διαχείρισης των HMS, βελτιώνοντας την συνολική απόδοση. Προφανώς, η δυνατότητα ακριβούς πρόβλεψης των μελλοντικών σελίδων μνήμης θα μπορούσε να βελτιώσει σημαντικά τη συνολική απόδοση αυτών των τεχνικών. Η πρόσφατη έξαρση του ενδιαφέροντος για τη μηχανική μάθηση έχει ωθήσει πολλούς ερευνητές στη διερεύνηση νέων εφαρμογών στην αρχιτεκτονική υπολογιστών, βασισμένων στη μηχανική μάθηση, προσφέροντας αποτελέσματα με μεγάλες προοπτικές για το μέλλον. Η παρούσα διπλωματική προτείνει μια νέα προσέγγιση, που αποτελείται από ένα καθορισμένο σύνολο κανόνων για την οπτική αναπαράσταση των δεδομένων και την αξιοποίηση μεθόδων μηχανικής μάθησης με βάση την εικόνα για την πρόβλεψη μελλοντικών προσβάσεων σελίδων. Η μέθοδος αυτή αποσκοπεί στην αξιοποίηση των πλεονεκτημάτων τόσο των χρονικών όσο και των χωρικών πληροφοριών που βρίσκονται εντός των δεδομένων. Βασικό στοιχείο αυτής της εργασίας είναι η αξιολόγηση της αποτελεσματικότητας αυτής της προσέγγισης σε σύγκριση με τις συμβατικές μεθόδους πρόβλεψης χρονοσειρών και τις τρέχουσες LSTM προσεγγίσεις καθώς και η εξερεύνηση του αχαρτογράφητου εδάφους της πρόβλεψης μεγαλύτερων τους ενός ακολουθιών μελλοντικών σελίδων. Αν και η προτεινόμενη προσέγγιση έχει προοπτικές εξέλιξης, δεν έχει ως στόχο να παρουσιάσει μια πρακτική υλοποίηση που να ξεπερνά ή να ανταγωνίζεται τις ήδη υπάρχουσες μεθόδους. Αντίθετα, θέτει τις βάσεις για την περαιτέρω ανάπτυξη τεχνικών μηχανικής μάθησης με βάση τις εικόνες για την πρόβλεψη σελίδων, που μπορούν να οδηγήσουν σε σημαντικές βελτιώσεις των επιδόσεων τόσο στα HMS όσο και στα συστήματα προφόρτωσης δεδομένων.

Λέξεις Κλειδιά: Πρότυπα Μνήμης, Βαθιά Νευρωνική Μάθηση, Όραση-Υπολογιστών, Πρόβλεψη χρονοσειρών, Μηχανική Μάθηση, Convolutional LSTM, Convolutional Autoencoder, Πρόβλεψη Σελίδας

Abstract

Modern computing systems, from personal devices to powerful data centers, have witnessed a significant rise in processing power. This remarkable progress, however, creates a growing disparity between compute power and memory access speed, leading to performance bottlenecks. On a hardware level, this translates to wasted compute cycles while waiting for data retrieval. On a system level, handling more information necessitates more memory. Given the limitations of Dynamic Random Access Memory (DRAM), Non-Volatile Memory (NVM) is introduced, creating Hybrid Memory Solutions (HMS). Computer architecture utilizes prediction methods to mitigate the mentioned problems. At the CPU level, this involves data prefetching, where relevant data is preloaded in the cache to mask memory latency. On a system level, various page-related techniques (scheduling, migration, replacement, etc.) are employed to optimize memory management for HMS, ultimately improving overall performance. Currently implemented solutions in both cases are not too complex and focus on hardware or low-level software solutions. Evidently, accurate future page prediction could significantly enhance their performance. Fueled by the recent surge in machine learning, researchers are exploring novel applications in computer architecture, uncovering promising solutions.

Inspired by image-based solutions for financial time series forecasting, this thesis proposes a new approach that leverages similar image-based machine learning models to predict future page accesses. A complete pipeline is proposed that consists of a defined set of rules to visually represent the data and utilize image-based machine learning methods to predict future page accesses. This method aims to harness the strengths of both temporal and spatial information within the data. The research seeks to evaluate the effectiveness of this approach compared to traditional timeseries forecasting methods and current state-of-the-art LSTMs, and to explore the uncharted territory of predicting longer sequences of future page accesses. Although the proposed approach is promising, it is not intended to provide a directly implementable solution that surpasses or competes with existing hardware-based methods. Instead, it paves the way and lays the groundwork for further development of image-based machine learning techniques for page forecasting, potentially leading to significant performance improvements in both HMS and data prefetching.

Key Words: Memory Patterns, Page Prediction, Timeseries Prediction, Machine Learning, Computer Vision, Deep Learning, Convolutional Autoencoder, Convolutional LSTM

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Παναγιώτη Τσανάκα για την εμπιστοσύνη που μου έδειξε για την εκπόνηση της εργασίας αυτής και για την καθοδήγηση σε όλη τη διάρκειά της. Θα ήθελα επίσης να ευχαριστήσω την κα Θάλεια-Δήμητρα Δούδαλη, Research Assistant Professor στο IMDEA Software Institute της Μαδρίτης, της οποίας η συνεχής βοήθεια και οι συμβουλές έπαιξαν καθοριστικό ρόλο στην τελική διαμόρφωση της παρούσας εργασίας. Είμαι ευγνώμων που μου έδωσε την ευκαιρία να κάνω την πρακτική μου άσκηση στο IMDEA, η οποία αποτέλεσε τη βάση για τη διαμόρφωση της διπλωματικής εργασίας μου, αλλά ήταν επίσης μια μοναδική εμπειρία εργασίας και γνώσης σε ερευνητικό περιβάλλον υψηλού επιπέδου.

Επιπλέον θα ήθελα να ευχαριστήσω τους γονείς και τα αδέρφια μου (και φυσικά το γάτο μου το Μπίμπο) για την υπομονή, την υποστήριξη και την εμπιστοσύνη που μου δείχνουν σε κάθε βήμα της σταδιοδρομίας μου. Τέλος, θέλω να ευχαριστήσω τους φίλους και συμφοιτητές μου, με τους οποίους μοιραστήκαμε αξέχαστες στιγμές και εμπειρίες όσο σπουδάζαμε μαζί και που πάντα με εμπύχωναν και με ωθούσαν να προσπαθώ για το καλύτερο.

Contents

Περίληψη	1
Abstract	2
Ευχαριστίες	3
Contents	4
List of Figures	6
List of Tables	9
0 Εκτεταμένη Ελληνική Περίληψη	11
0.1 Εισαγωγή	11
0.2 Σχετική Βιβλιογραφία	12
0.3 Θεωρητικό Υπόβαθρο	13
0.4 Δεδομένα και Οπτικοποίηση/Προεπεξεργασία	13
0.5 Πρόβλεψη με Μεθόδους Μηχανικής Μάθησης Βασισμένες σε Εικόνες	19
0.6 Σύγκριση με Άλλες Τεχνικές Πρόβλεψης Χρονοσειρών	21
0.7 Συμπεράσματα και Μελλοντικές Προεκτάσεις	26
1 Introduction	28
1.1 Motivation	28
1.2 Related Work	29
1.2.1 Page Prediction	30
1.3 Thesis Outline	32
2 Background	34
2.1 Traditional Forecasting Methods	34
2.1.1 Exponential Smoothing	34
2.1.2 ARIMA	35
2.2 Deep Learning	36
2.2.1 Recurrent Neural Networks and Long Short-Term Memory Networks	36
2.2.2 Convolutional Neural Networks	38
2.2.3 Convolutional Autoencoders	39
2.2.4 Convolutional Long Short-Term Memory Networks	39
3 Visualization of Data Access Patterns	41
3.1 Dataset Description	41
3.1.1 Dataset Source	41
3.1.2 Raw Data Format	43

3.2	Image Based Pipeline	44
3.2.1	Input Transformation/Data Preprocessing	44
3.2.2	Data Visualization	47
3.2.3	Image Decomposition	55
3.2.4	Image Dataset	56
4	Forecasting With Image-based Machine Learning Methods	59
4.1	Experimental Methodology	59
4.2	Data Splitting	60
4.3	Deployment of the Image Based Deep Learning Models	61
4.4	Evaluation Metrics	65
4.5	Tuning of the Forecasting History	70
4.6	Tuning of the Image Height Size	72
4.7	Tuning of the Forecasting Horizon	75
4.8	Chapter Summary	77
5	Comparison Against <i>LSTM</i> Forecasting Method	79
5.1	Traditional Time-series Forecasting Methods	79
5.2	Current State-Of-The-Art Methods Proposed	81
5.3	Comparison on History	84
5.4	Comparison on Forecasting Horizon	85
5.5	Comparison on Training and Inference Times	89
5.6	Comparison Throughout the Dataset	91
5.7	Chapter Summary	94
6	Conclusions and Future Work	96
6.1	Conclusion	96
6.2	Challenges and Limitations	97
6.3	Discussion and Future Direction	98

List of Figures

0.1	Η εσωτερική δομή του <i>ConvLSTM</i> [74].	13
0.2	Παράδειγμα για τις 2 πρώτες τιμές διεύθυνσης ενός trace αρχείου.	15
0.3	Γραφήματα με την χρήση <code>matplotlib</code> για (a) bc-0 (b) bfs-10 (c) 410.bwaves-s0 (d) 433.milc-s1 (e) 605.mcf-s8 (f) cc-5 trace αρχεία.	16
0.4	Παράδειγμα εστίασης σε περιοχή Black Bar στο bc-0 benchmark.	17
0.7	Μετατροπή εικόνας πίσω σε χρονοσειρά.	18
0.5	Παράδειγμα μετατροπής μιας τιμής της χρονοσειράς σε εικόνα μεγέθους 1x10.	18
0.6	Απλοποιημένο διάγραμμα που απεικονίζει την διαδικασία οπτικοποίησης της χρονοσειράς εισόδου σε εικόνες μεγέθους 10x10.	18
0.8	Είσοδος και αναμενόμενη πρόβλεψη για $fh=6$ and $fh=4$	19
0.9	Αρχιτεκτονική Μοντέλου <i>ConvLSTM-One-Img</i> . Οι τιμές και οι διαστάσεις που εμφανίζονται προέρχονται από ένα τυχαίο παράδειγμα, καθώς εξαρτώνται από το μέγεθος της εικόνας εισόδου.	20
0.10	Παράδειγμα απεικόνισης προβλέψεων με Exponential Smoothing και ARIMA μαζί με την πραγματικές τιμές, για ορίζοντα πρόβλεψης $fh = 32$. Σύνολο 15 προβλέψεων έχουν συνενωθεί για κάθε μία από τις εικόνες.	22
0.11	Σύγκριση <i>ConvLSTM-One-Img</i> και <i>LSTM</i> μοντέλων. (a) Σύγκριση των τιμών Accuracy για διαφορετικές τιμές ιστορίας πρόβλεψης, (b) Σύγκριση των τιμών Accuracy για διαφορετικές τιμές ορίζοντα πρόβλεψης (έχουν συμπεριληφθεί οι στατιστικές μεθόδοι Exponential Smoothing και ARIMA) και (c) Σύγκριση του Inference Time για διαφορετικές τιμές ιστορίας πρόβλεψης	24
0.12	Παράδειγμα απεικόνισης προβλέψεων <i>ConvLSTM-One-Img</i> ($Accuracy=0.36$) και <i>LSTM</i> ($Accuracy=0.30$) και των πραγματικών τιμών, για ορίζοντα πρόβλεψης $fh = 32$ και ιστορία πρόβλεψης 128. Σύνολο 16 προβλέψεων έχουν συνενωθεί για κάθε μία από τις εικόνες	24
0.13	Οπτική απεικόνιση της διαφοράς στην "πυκνότητα" των μοτίβων (strides) σε διαφορετικά σημεία στο ίδιο benchmark. Τα 2 κόκκινα εσιασμένα (zoomed-in) παράθυρα είναι ίδιου μεγέθους Δcc	25
0.14	Σύγκριση <i>ConvLSTM-One-Img</i> και <i>LSTM</i> μοντέλων για εκπαίδευση στο 10-30% του dataset και έλεγχο ανα 10% για το υπόλοιπο dataset.	26
2.1	Neural Networks Core Structure	36
2.2	The different types of Recurrent Neural Networks.	37
2.3	The different types of Recurrent Neural Networks.	37
2.4	The application of a convolutional kernel within a CNN architecture.	38
2.5	<i>Sigmoid</i> and <i>Relu</i> activation functions.	38
2.6	High-level architecture of an autoencoder.	39
2.7	Inner structure of <i>ConvLSTM</i> [74].	40

3.1	Plot of the total number of load instructions of all benchmarks (a) scatter plot of the number of load instructions for each benchmark (b) box plot of the number of load instructions for each benchmark (c) scatter plot of the number of <i>miss</i> load instructions for each benchmark (d) box plot of the number of <i>miss</i> load instructions for each benchmark	45
3.2	Plot depicting the hits and misses of load instruction for each benchmark in each suite. (a) for GAP benchmark (b) for SPEC 2006 and (c) for SPEC 2017.	47
3.3	Plot depicting distribution of the total number of different pages accessed by all benchmarks	47
3.4	Pipeline example for the first 2 inputs for a new trace file execution.	48
3.5	Plots using <code>matplotlib</code> for (a) bc-0 (b) bfs-10 (c) 410.bwaves-s0 (d) 433.milc-s1 (e) 605.mcf-s8 (f) cc-5 trace files	50
3.6	Examples of <i>zooming</i> on benchmarks (a) 605.mcf-s1 (b) cactuBSSN-s0 . . .	51
3.7	Examples of <i>zooming</i> on benchmarks (a) 473.astar-s1 (b) 473.astar-s2	52
3.8	Example of transforming a data point to a corresponding image of 1x10 size	54
3.9	Simplified diagram depicting the visualization process of the input time-series to 10x10 images.	55
3.10	Decomposition of each image to a list of pairs of interest.	56
3.11	Input and label example for $fh=6/overlap=40\%$ and $fh=4/overlap=60\%$. . .	57
3.12	Input and label of different size example for $fh=6$ and $fh=4$	58
3.13	Multiple input images example	58
4.1	Visualized difference of "density" of stride-like patterns within the same benchmark. In this visualization the Δcc is the same between the two zoomed-in images (images within the red bounding boxes)	61
4.2	Sliding window cross validation example. Image Source: www.r-bloggers.com	61
4.3	<i>ConvLSTM-Mult-Img</i> Model Architecture. The values and dimensions appeared are from a random example as they are dependent from the size of the input image.	63
4.4	<i>ConvLSTM-One-Img</i> Model Architecture. The values and dimensions appeared are from a random example as they are dependent from the size of the input image.	64
4.5	Prediction Images before (left) and after (right) they are "Binarized". The brightest pixel of each column is kept and rounded to a value of 1.	65
4.6	Plots using <code>matplotlib</code> for metrics results shown on Tables 4.1 and 4.2: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) $RMSE$ (d) IoU (e) $MAPE$ (f) <i>Inference Time</i> for 1 prediction.	72
4.7	Plots using <code>matplotlib</code> for metrics results shown on Tables 4.3 and 4.4: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) $RMSE$ (d) IoU (e) $MAPE$.	74
4.8	Plots using <code>matplotlib</code> for metrics results shown on Tables 4.5: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) $RMSE$ (d) IoU (e) $MAPE$. . .	77
5.1	Example plots of <i>Exponential Smoothing</i> and <i>ARIMA</i> predictions with the ground truth for a $fh = 32$. A total of 15 predictions are concatenated together for the each of the images	81
5.2	<i>LSTM</i> Model Architecture. The values and dimensions are affected by the input and output size of the model. In this case the <i>forecasting horizon</i> is 32 and <i>forecasting History</i> is 128.	83
5.3	Plots using <code>matplotlib</code> for performance metrics results shown on Table 5.3 for <i>LSTM</i> model: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) $RMSE$ (d) IoU (e) $MAPE$ (f) <i>Inference Time</i>	85

5.4	Plots using <code>matplotlib</code> for metrics results shown on Tables 4.5, 4.7, 5.1 and 5.2 comparing the <i>Exponential Smoothing</i> , <i>ARIMA</i> , <i>LSTM</i> and <i>ConvLSTM-One-Img</i> models: (a) <i>Accuracy/Dice Coefficient</i> (b) <i>MAE_{scaled}</i> (c) <i>RMSE</i> (d) <i>IoU</i> (e) <i>MAPE</i>	88
5.5	Example plots of <i>ConvLSTM-One-Img</i> (<i>Accuracy</i> =0.36) and <i>LSTM</i> (<i>Accuracy</i> =0.30) predictions compared with the <i>Ground Truth</i> for a <i>fh</i> = 32 and a history of 128 data points. A total of 16 predictions are concatenated together for the each of the images creating an image of $64 \times 512 \times 1$ image.	88
5.6	Example plots of <i>ConvLSTM-One-Img</i> (<i>Accuracy</i> =0.48) and <i>LSTM</i> (<i>Accuracy</i> =0.16) predictions compared with the <i>Ground Truth</i> for a <i>fh</i> = 32 and a history of 128 data points. A total of 16 predictions are concatenated together for the each of the images creating an image of $64 \times 512 \times 1$ image.	89
5.7	Loss function plots for (a) <i>LSTM</i> and (b) <i>ConvLSTM-One-Img</i> model.	91
5.8	Plots using <code>matplotlib</code> for metrics results shown on Tables 5.6, 5.7 and 5.8 for <i>LSTM</i> and <i>ConvLSTM-One-Img</i> models with a forecasting history of 128 (a) <i>Accuracy/Dice Coefficient</i> (b) <i>MAE_{scaled}</i> (c) <i>RMSE</i> (d) <i>IoU</i> (e) <i>MAPE</i>	93

List of Tables

0.1	Πρώτες πέντε γραμμές του αρχείου trace bc-0.txt. Όπου οι διαχωρισμένες με κόμμα τιμές κάθε γραμμής αντιστοιχούν σε μια οδηγία πρόσβασης της κύριας μνήμης και οι τιμές από αριστερά προς τα δεξιά είναι: το Μοναδικό Αναγνωριστικό Διαδικασίας της φόρτωσης, ο χρονικός κύκλος της CPU, η φυσική διεύθυνση του αιτήματος μνήμης, Μετρητής Προγράμματος (PC) και επιτυχία (1) ή αποτυχία (0) της κρυφής μνήμης τελευταίου επιπέδου (LLC).	14
0.2	Το διάστημα σελίδων που αντιστοιχεί το <i>Black Bar</i> κομμάτι για κάθε ένα από τα benchmarks, το σύνολο των αστοχιών εντός αυτού του διαστήματος σελίδων καθώς και το ποσοστό επί του συνόλου των αστοχιών	17
3.1	Example of benchmarks included in SPEC2017 suite and a brief explanation of application area	42
3.2	Graph Kernels of GAP Benchmark Suite	43
3.3	Input graphs of GAP Benchmark Suite. $ V $ is number of vertices and $ E $ number of Edges of graph	43
3.4	First five rows of bc-0.txt trace file. Where the comma separated values of each row correspond to a load memory access instruction and the values from left to right are the Unique Instruction Id of the load, CPU cycle timestamp, Physical address of the memory request, Instruction Pointed of the load (PC) and Last Level Cache (LLC) hit/miss (1/0)	44
3.5	Page range of <i>Black Bar</i> part of benchmarks, total number of misses in given page-range and their coverage compared to all misses of benchmark	53
4.1	Performance metrics of <i>ConvLSTM-Mult-Img</i> model for different numbers of input images ($64 \times 64 \times 1$) and $fh = 64$	71
4.2	Performance metrics of <i>ConvLSTM-One-Img</i> model for input images of different x-axis size ($64 \times X' \times 1$), as forecasting history changes for the <i>ConvLSTM-One-Img</i> model for $fh = 64$	71
4.3	Performance metrics of <i>ConvLSTM-One-Img</i> model for input image of ($Y' \times 128 \times 1$) and $fh = 64$	73
4.4	Performance metrics of <i>ConvLSTM-One-Img</i> model for input image of ($Y' \times 256 \times 1$) and $fh = 64$	73
4.5	Performance metrics of <i>ConvLSTM-One-Img</i> model for input image of ($64 \times 128 \times 1$) and different forecasting horizons	76
5.1	Performance metrics of <i>Exponential Smoothing</i> for different values of forecasting horizon	80
5.2	Performance metrics of <i>ARIMA</i> for different values of forecasting horizon. We have "-" on the total time for inference of the model for the forecasting horizons of 16 and less because a smaller testing dataset (equally distributed through the dataset) was given due to the great time needed for inference	80

5.3	Performance metrics as forecasting history changes for the <i>LSTM</i> model for $fh = 64$	83
5.4	Performance metrics as forecasting horizon changes for the <i>LSTM</i> model with a forecasting history of 128 as input.	84
5.5	Training time, model size and number of trainable parameters for <i>ConvLSTM-One-Img</i> and <i>LSTM</i> model. Both models forecasting history is 128.	90
5.6	<i>Accuracy</i> performance on different parts of the dataset	92
5.7	MAE_{scaled} and <i>RMSE</i> metrics performance on different parts of the dataset	92
5.8	<i>IoU</i> and <i>MAPE</i> metrics performance on different parts of the dataset	92
5.9	Performance metrics of <i>LSTM</i> and <i>ConvLSTM</i> models trained and tested on different parts of the dataset. Forecasting History is 128 and Forecasting Horizon 32.	94

Chapter 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Τα σύγχρονα υπολογιστικά συστήματα, από τις προσωπικές συσκευές έως τα ισχυρά κέντρα δεδομένων, έχουν σημειώσει μεγάλη πρόοδο στην απόδοσή τους, ως αποτέλεσμα της σημαντικής αύξησης στην επεξεργαστική τους ισχύ. Αυτή η αξιοσημείωτη πρόοδος, ωστόσο, αυξάνει την ανισότητα μεταξύ της υπολογιστικής ισχύος και της ταχύτητας πρόσβασης στη μνήμη, οδηγώντας σε μείωση της δυναμικής απόδοσης [88]. Έχει παρατηρηθεί ότι σύγχρονες εφαρμογές μπορούν να περάσουν σε ορισμένες περιπτώσεις σχεδόν το μισό του χρόνου εκτέλεσής τους σε αυτό το στάδιο [43, 33]. Σε επίπεδο επεξεργαστή, αυτό μεταφράζεται σε χαμένους υπολογιστικούς κύκλους κατά την αναμονή για την ανάκτηση δεδομένων. Σε επίπεδα συστήματος, η δυνατότητα επεξεργασίας περισσότερων δεδομένων δημιουργεί την ανάγκη για αύξηση της απαιτούμενης μνήμης. Λόγω των περιορισμών κλιμάκωσης της συνολικής δυναμικής μνήμης τυχαίας προσπέλασης (Dynamic Random Access Memory, DRAM), γίνεται η χρήση μη πτητικής μνήμης (Non-Volatile Memory, NVM), δημιουργώντας υβριδικές λύσεις μνήμης (Hybrid Memory Systems, HMS) [85, 22, 44].

Η αρχιτεκτονική υπολογιστών χρησιμοποιεί μεθόδους πρόβλεψης για τον μετριασμό των προαναφερθέντων προβλημάτων. Στο επίπεδο του επεξεργαστή, αυτό περιλαμβάνει την προφόρτωση δεδομένων (data prefetching), όπου τα δεδομένα που θα χρειαστούν σε επόμενο χρόνο φορτώνονται εκ των προτέρων στην κρυφή μνήμη cache για να ελαττώσουν τις καθυστερήσεις φόρτωσης δεδομένων από την μνήμη. Σε επίπεδο συστήματος, χρησιμοποιούνται διάφορες τεχνικές που σχετίζονται με τις σελίδες (χρονοπρογραμματισμός, μεταφορά, αντικατάσταση κ.λπ.) για τη βελτιστοποίηση της διαχείρισης της μνήμης των HMS, βελτιώνοντας την συνολική απόδοση. Οι τρέχουσες διαθέσιμες μέθοδοι για την εφαρμογή των παραπάνω τεχνικών είναι σχετικά απλές και επικεντρώνονται σε λύσεις υλικού (hardware) ή λογισμικού χαμηλού επιπέδου [38, 9, 54]. Προφανώς, η δυνατότητα ακριβούς πρόβλεψης των μελλοντικών σελίδων θα μπορούσε να βελτιώσει σημαντικά τη συνολική απόδοση αυτών των τεχνικών. Η πρόσφατη έξαρση του ενδιαφέροντος για τη μηχανική μάθηση έχει ωθήσει πολλούς ερευνητές στη διερεύνηση νέων εφαρμογών στην αρχιτεκτονική υπολογιστών, βασισμένων στη μηχανική μάθηση, προσφέροντας αποτελέσματα με μεγάλες προοπτικές για το μέλλον.

Οι εξελισσόμενες δυνατότητές του επιτρέπουν ισχυρές και ακριβείς προβλέψεις, ιδιαίτερα για ακανόνιστα μοτίβα πρόσβασης στη μνήμη. Η μηχανική μάθηση μπορεί επίσης να βοηθήσει σε προβλέψεις πιο "μακριά" στο μέλλον (long-term forecasting) [71, 61], παρέχοντας περισσότερη πληροφορία για τις επερχόμενες σελίδες. Έρευνα έχει εκπονηθεί με την χρήση διαφορετικών αλγορίθμων μηχανικής μάθησης προκειμένου να βελτιωθεί η απόδοση των αναφερόμενων συστημάτων. Αυτά περιλαμβάνουν τη χρήση μεθόδων Recurrent Neural Network (RNNs) [78,

75, 44], Natural Language Processing (NLP) [33], Transformers [90] και Reinforcement Learning [22, 6]. Πρέπει να σημειωθεί ότι οι περισσότερες από τις προτεινόμενες μέθοδοι, ειδικά στο προανάκτηση δεδομένων στην cache (data prefetching), παρόλο που παρέχουν εντυπωσιακές επιδόσεις, δεν είναι ακόμα εφικτές για πρακτική εφαρμογή λόγω περιορισμών σε μέγεθος, υπολογιστικές απαιτήσεις χρόνο απόκρισης (Inference Time). Συνεπώς βρίσκονται ακόμα σε θεωρητικό και διερευνητικό στάδιο και ακόμα εξετάζεται ιδιαίτερα η πρακτική τους υλοποίηση.

Ανάγουμε το πρόβλημα της πρόβλεψης μελλοντικών σελίδων μνήμης σε αυτό της το πρόβλεψης χρονοσειρών. Αξιοποιώντας υπάρχουσες μεθοδολογίες πρόβλεψης χρηματοοικονομικών χρονοσειρών με την χρήση εικόνων [19], η παρούσα διπλωματική προτείνει μια νέα προσέγγιση που αποτελείται από ένα καθορισμένο σύνολο κανόνων για την οπτική αναπαράσταση των δεδομένων και την αξιοποίηση μεθόδων μηχανικής μάθησης με βάση την εικόνα για την πρόβλεψη μελλοντικών προσβάσεων σελίδων. Η μέθοδος αυτή αποσκοπεί στην αξιοποίηση των πλεονεκτημάτων τόσο των χρονικών όσο και των χωρικών πληροφοριών που βρίσκονται εντός των δεδομένων. Βασικό στοιχείο αυτής της εργασίας είναι η αξιολόγηση της αποτελεσματικότητας αυτής της προσέγγισης σε σύγκριση με τις συμβατικές μεθόδους πρόβλεψης χρονοσειρών και τις τρέχουσες LSTM προσεγγίσεις καθώς και η εξερεύνηση του αχαρτογράφητου εδάφους της πρόβλεψης μεγαλύτερων τους ενός ακολουθιών μελλοντικών σελίδων. Συμπωματικά, μια πολύ πρόσφατη δημοσίευση που προτάθηκε κατά την διάρκεια διεξαγωγή αυτής της διπλωματικής, προτείνει ένα μοντέλο για πρόβλεψη διευθύνσεων μνήμης με την χρήση εικόνων με αποτελέσματα που ξεπερνούν προηγούμενες εργασίες, επικυρώνοντας έτσι την προσέγγιση που ακολουθήσαμε. [57].

Αν και η προτεινόμενη προσέγγιση έχει προοπτικές εξέλιξης, δεν έχει ως στόχο να παρουσιάσει μια πρακτική υλοποίηση που να ξεπερνά ή να ανταγωνίζεται τις ήδη υπάρχουσες ήδη μεθόδους. Αντίθετα, θέτει τις βάσεις για την περαιτέρω ανάπτυξη τεχνικών μηχανικής μάθησης (Machine Learning, ML) με βάση τις εικόνες για την πρόβλεψη σελίδων, που μπορούν να οδηγήσουν σε σημαντικές βελτιώσεις των επιδόσεων τόσο στα HMS όσο και στα συστήματα προφόρτωσης δεδομένων.

0.2 Σχετική Βιβλιογραφία

Σε αυτήν την ενότητα, παρουσιάζεται μια σύνοψη της προηγούμενης ερευνητικής εργασίας που σχετίζεται με τα θέματα ενδιαφέροντος της παρούσας μελέτης. Θα παρουσιαστούν λύσεις που βασίζονται στην μηχανική μάθηση.

Ένα πρωτοποριακό έργο από τους Hashemi et al. [33] χρησιμοποιεί ιδέες από την επεξεργασία φυσικής γλώσσας (NLP) και μοντέλα RNN για πρόβλεψη διευθύνσεων μνήμης. Το MemMAP από Srivastava et al. εισάγει μικρά, αποδοτικά και γενικευμένα μοντέλα LSTM για ακριβή πρόβλεψη της επόμενης πρόσβασης μνήμης. Πιο πρόσφατα, το TransforMAP από Zhang et al. αξιοποιεί το ισχυρό μοντέλο Transformer για πρόβλεψη πολλών γραμμών της cache.

Μια μεγάλη δυσκολία του που παρουσιάζεται στο πρόβλημα της πρόβλεψης διευθύνσεων μνήμης είναι η εκθετική αύξηση των πιθανών αποτελεσμάτων. Μια λύση δίνεται από την χωρισμό σε : (α) Πρόβλεψη σελίδας λειτουργικού συστήματος και (β) Πρόβλεψη μετατόπισης (offset) εντός μιας σελίδας. Το Voyager από Shi et al. [75] αποτελεί ένα από τα προτεινόμενα σύστημα μηχανικής μάθησης με τις καλύτερες επιδόσεις, που αποδεικνύει ότι τα LSTM μπορούν και υπερτερούν των υφιστάμενων μεθόδων.

Τέλος, το Drishyam από Mohapatra et al. [57] αξιοποιεί τεχνικές όρασης υπολογιστή για πρόβλεψη του μέλλοντος με προσέγγιση ταξινόμησης (classification) εικόνων. Τα αποτελέσματα δείχνουν υψηλή ακρίβεια και ευρύτερο φάσμα σεναρίων σε σύγκριση με το Voyager.

0.3 Θεωρητικό Υπόβαθρο

Στην ενότητα αυτή θα παρουσιάσουμε το απαραίτητο υπόβαθρο για την εξήγηση και την πλήρη κατανόηση μοντέλου που πρωτάφηκε και χρησιμοποιήθηκε στο κύριο μέρος της διπλωματικής. Τα Συνελικτικά Δίκτυα Μακράς Βραχυπρόθεσμης Μνήμης (Convolutional Long Short-Term Memory networks, ConvLSTMs) είναι μια εξειδικευμένη μορφή Δικτύων Επαναλαμβανόμενων Στοιχείων (RNNs) σχεδιασμένα να χειρίζονται διαδοχικά δεδομένα (sequential data) με χωρική δομή, όπως βίντεο ή δεδομένα χρονοσειρών. Σε αντίθεση με τα παραδοσιακά LSTMs που βασίζονται αποκλειστικά στην διαδοχική ανάλυση δεδομένων, τα ConvLSTMs ενσωματώνουν πράξεις συνελίξης για να αναλύσουν ταυτόχρονα συνεχόμενα καρέ ή τμήματα, καταγράφοντας έτσι τις χωρικές και χρονικές εξαρτήσεις μέσα στα δεδομένα. Αυτό επιτρέπει στα ConvLSTMs να μοντελοποιούν αποτελεσματικά σύνθετα μοτίβα και δυναμικές που υπάρχουν σε διαδοχικά δεδομένα, αξιοποιώντας τις εγγενείς χωρικές σχέσεις μεταξύ σειρών καρέ.

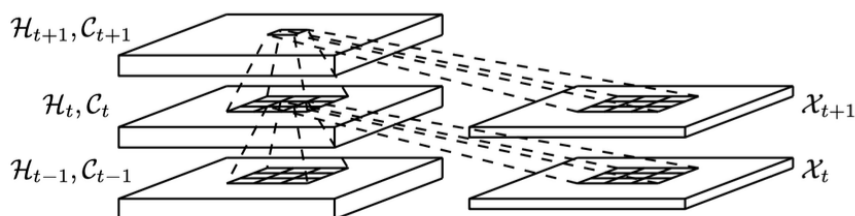


Figure 0.1: Η εσωτερική δομή του *ConvLSTM* [74].

Για χωροχρονική πρόβλεψη, το ConvLSTM επεξεργάζεται την μελλοντική κατάσταση ενός κελιού πλέγματος βάσει τις εισόδους και των ιστορικών καταστάσεων των γειτονικών του περιοχών. Μια απλή μέθοδος για να γίνει αυτό είναι η χρήση ενός τελεστή συνελίξης στις μεταβάσεις εισόδου-προς-κατάσταση (input-to-state) και κατάσταση-προς-κατάσταση (state-to-state) Εικόνα (Figure) 0.1 [74]. Με άλλα λόγια, λειτουργεί ως επαναλαμβανόμενο στρώμα με τον ίδιο τρόπο όπως το LSTM, εκτός από το ότι αντί για εσωτερικούς πολλαπλασιασμούς πινάκων, χρησιμοποιούνται πράξεις συνελίξης.

0.4 Δεδομένα και Οπτικοποίηση/Προεπεξεργασία

Δεδομένα

Τα δεδομένα προέρχονται από τον "Διαγωνισμό Προφόρτωσης Μηχανικής Μάθησης" (2021) που έγινε κατά τη διάρκεια του Εργαστηρίου Μηχανικής Μάθησης για Αρχιτεκτονική και Συστήματα Υπολογιστών (MLArchSys) [2], που πραγματοποιείται στο International Symposium on Computer Architecture (ISCA). Το σύνολο δεδομένων παρέχει πληροφορίες της εκτέλεσης διαφόρων απαιτητικά σε μνήμη benchmarks, με στόχο την αξιολόγηση μοντέλων μηχανικής μάθησης για προφόρτωση δεδομένων. Τα αρχεία ιχνηλασίας (traces) αποτελούνται από μια ακολουθία φορτώσεων μνήμης σε επίπεδο 64 byte που συλλέγονται στην κρυφή μνήμη τελευταίου επιπέδου (Last Level Cache, LLC), στο επίπεδο L3 cache στην περίπτωση μας, επομένως η ροή φιλτράρεται από την μνήμη cache L2. Τα benchmark προέρχονται από τα SPEC 2006 [3, 34], SPEC 2017 [4, 15] και GAP [13] σουίτες.

Τα αρχεία traces παρέχονται σε μορφή απλού κειμένου διαχωρισμένου με κόμμα (comma separated file, ".txt"), όπου κάθε γραμμή αντιπροσωπεύει μια εντολή πρόσβασης στη μνήμη και αντιστοιχεί στις ακόλουθες πληροφορίες:

- Μοναδικό αναγνωριστικό οδηγίας (Unique Instruction Id): Αυτό χαρακτηρίζει μοναδικά την οδηγία φόρτωσης.
- Χρονική σήμανση κύκλου CPU για τη φόρτωση (CPU cycle timestamp): Αυτή η τιμή υποδεικνύει σε ποιον κύκλο CPU εκτελέστηκε η πρόσβαση φόρτωσης.
- Φυσική διεύθυνση του αιτήματος μνήμης (Physical address of the memory request): Αυτή η τιμή καθορίζει τη διεύθυνση στη φυσική μνήμη που ζητήθηκε να φορτωθεί.
- Μετρητής Προγράμματος (Program Counter, PC): Αυτή η τιμή υποδεικνύει τη διεύθυνση της οδηγίας φόρτωσης στον κώδικα του προγράμματος.
- Επιτυχία (Hit, 1) ή Αποτυχία (Miss, 0) στην κρυφή μνήμη τελευταίου επιπέδου (LLC): Η τιμή "1" υποδεικνύει ότι τα δεδομένα βρέθηκαν στην κρυφή μνήμη τελευταίου επιπέδου (LLC), ενώ η τιμή "0" υποδεικνύει ότι τα δεδομένα δεν βρέθηκαν στην LLC και έπρεπε να ανακτηθούν από την κύρια μνήμη.

Ένα παράδειγμα των πρώτων 5 γραμμών ενός από τα αρχεία φαίνεται στον Πίνακα (Table) 0.1.

5 πρώτες γραμμές του bc-0.txt				
14,	152,	28e837c89f00,	14ce07e1e230,	0
17,	169,	28e837c8af40,	14ce07e1e240,	0
18,	509,	28e837c8c840,	14ce07e1e247,	0
19,	633,	28e837c8c740,	14ce07e1e24b,	0
22,	634,	28e837c8c7c0,	14ce07e1e254,	0

Table 0.1: Πρώτες πέντε γραμμές του αρχείου trace bc-0.txt. Όπου οι διαχωρισμένες με κόμμα τιμές κάθε γραμμής αντιστοιχούν σε μια οδηγία πρόσβασης της κύριας μνήμης και οι τιμές από αριστερά προς τα δεξιά είναι: το Μοναδικό Αναγνωριστικό Διαδικασίας της φόρτωσης, ο χρονικός κύκλος της CPU, η φυσική διεύθυνση του αιτήματος μνήμης, Μετρητής Προγράμματος (PC) και επιτυχία (1) ή αποτυχία (0) της κρυφής μνήμης τελευταίου επιπέδου (LLC).

Προεπεξεργασία

Η διαχείριση μνήμης σε σελίδες αποτελεί μια από τις πιο γνωστές τεχνικές των λειτουργικών συστημάτων (Operating Systems, OS). Από τη δεκαετία του '60 έως και σήμερα, σχεδόν όλα τα OS χρησιμοποιούν σελίδες μνήμης μεγέθους 4 KB [86], συνεπώς και εμείς θα υποθέσουμε σελίδα ίση με 4 KB. Τα trace αρχεία που δίνονται παρέχουν τη φυσική διεύθυνση ως αριθμό σε δεκαεξαδικό σύστημα, η οποία αποτελεί την τρίτη τιμή σε κάθε γραμμή του Πίνακα (Table) 0.1. Λαμβάνουμε υπόψη, μόνο τις περιπτώσεις που υπήρξε αποτυχία στην εύρεση των δεδομένων στην LLC, δηλαδή της περιπτώσεις miss (τιμή 0 στην τελευταία στήλη του trace αρχείου). Απο το παραπάνω, έχουμε μια ακολουθία $\{h_0, h_1, \dots, h_t\}$ όπου h_t είναι ένας δεκαεξαδικός αριθμός. Παίρνουμε κάθε δεκαεξαδική διεύθυνση h_t και τη μετατρέπουμε στην αντίστοιχη δεκαδική τιμή, το οποίο μας δίνει μια ακολουθία $\{d_0, d_1, \dots, d_t\}$ όπου d_t είναι ένας ακέραιος αριθμός βάσης 10. Στη συνέχεια, παίρνουμε κάθε διεύθυνση d_t και βρίσκουμε τον πλησιέστερο ακέραιο αριθμό διαιρέσιμο με το 4096 (το μέγεθος σελίδας που επιλέξαμε, δηλαδή 4 KB), δίνοντάς μας τη διεύθυνση βάσης dp_t . Με αυτόν τον τρόπο, αντιστοιχούμε ένα εύρος δεκαεξαδικών διευθύνσεων σε δεκαδική διεύθυνση. Στη συνέχεια, χρησιμοποιούμε αυτές τις διευθύνσεις dp_t ως κλειδί (key) για να αντιστοιχίσουμε (map) καθεμία τους σε

έναν ακέραιο αριθμό y_t ξεκινώντας από το μηδέν και αυξάνοντας κατά 1 με κάθε νέο κλειδί-διεύθυνση που συναντάμε. Αν συναντήσουμε μια διεύθυνση dp_t όπου $dp_t = dp_{t-1}$, τότε με βάση την αντιστοίχιση (mapping) έχουμε $y_t = y_{t-1}$. Ένα παράδειγμα της παραπάνω διαδικασίας φαίνεται στην Εικόνα (Figure) 3.4. Επαναλαμβάνοντας τα παραπάνω για όλες τις τιμές miss τιμές $\{h_0, h_1, \dots, h_t\}$ ενός trace αρχείου, καταλήγουμε σε μια μονοδιάστατη (1D) χρονοσειρά ακέραιων τιμών $\{y_0, y_1, \dots, y_t\}$ όπου $y_t \in \mathbb{Z}^+$.

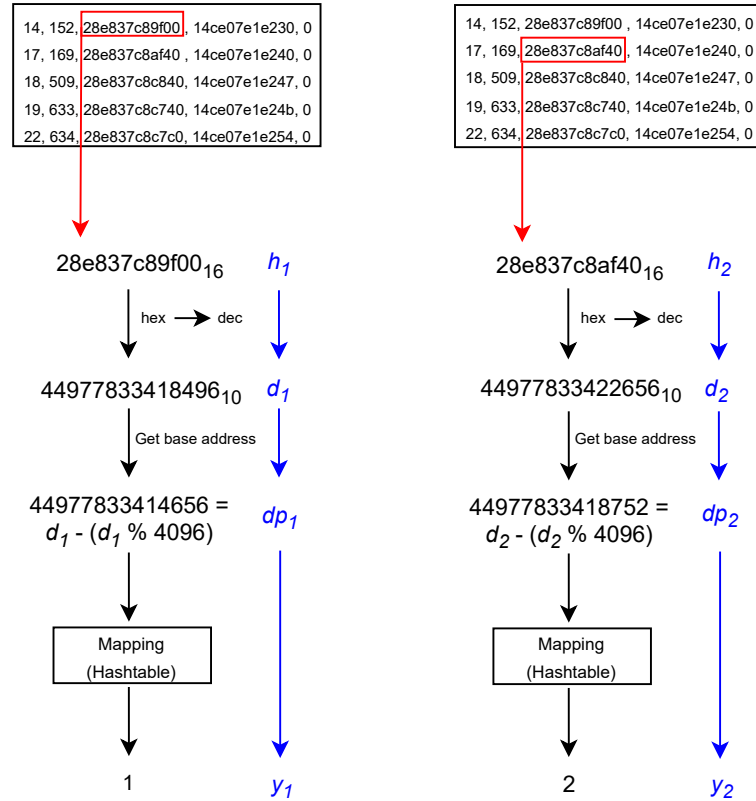


Figure 0.2: Παράδειγμα για τις 2 πρώτες τιμές διεύθυνσης ενός trace αρχείου.

Οπτικοποίηση

Βασισμένοι στην εργασία των Veloso et al. [19] θα έχουμε μια απλή προσέγγιση για την απεικόνιση της χρονοσειράς. Όπως αναφέρθηκε προηγουμένως, η χρονοσειρά που δημιουργήσαμε προηγουμένως $\{y_0, y_1, \dots, y_t\}$ όπου $y_t \in \mathbb{Z}^+$, μπορεί να έχει μήκος στις τάξεις μεγέθους των εκατομμυρίων. Αρχικά, για τη δημιουργία γραφήματος της παραπάνω χρονοσειράς, χρησιμοποιούμε τη βιβλιοθήκη `python matplotlib`, η οποία κλιμακώνει αυτόματα την είσοδο της σειράς σε μια εικόνα συγκεκριμένου μεγέθους. Ορισμένα παραδείγματα γραφημάτων (ολόκληρου του συνόλου δεδομένων) μπορούν να προβληθούν στο Γράφημα (Figure) 0.3. Επιπλέον, πρέπει να σημειωθεί ότι δημιουργήθηκαν δυαδικές εικόνες (δηλαδή μονο μαύρα ή άσπρα pixel). Όπως φαίνεται από τα παραδείγματα γραφημάτων του Σχήμα 0.3, ορισμένα γραφήματα δείχνουν επαναλαμβανόμενα μοτίβα *strides*, δηλαδή σελίδες που επαναλαμβάνονται συνεχόμενα καθ' όλη τη διάρκεια της εκτέλεσης του benchmark.

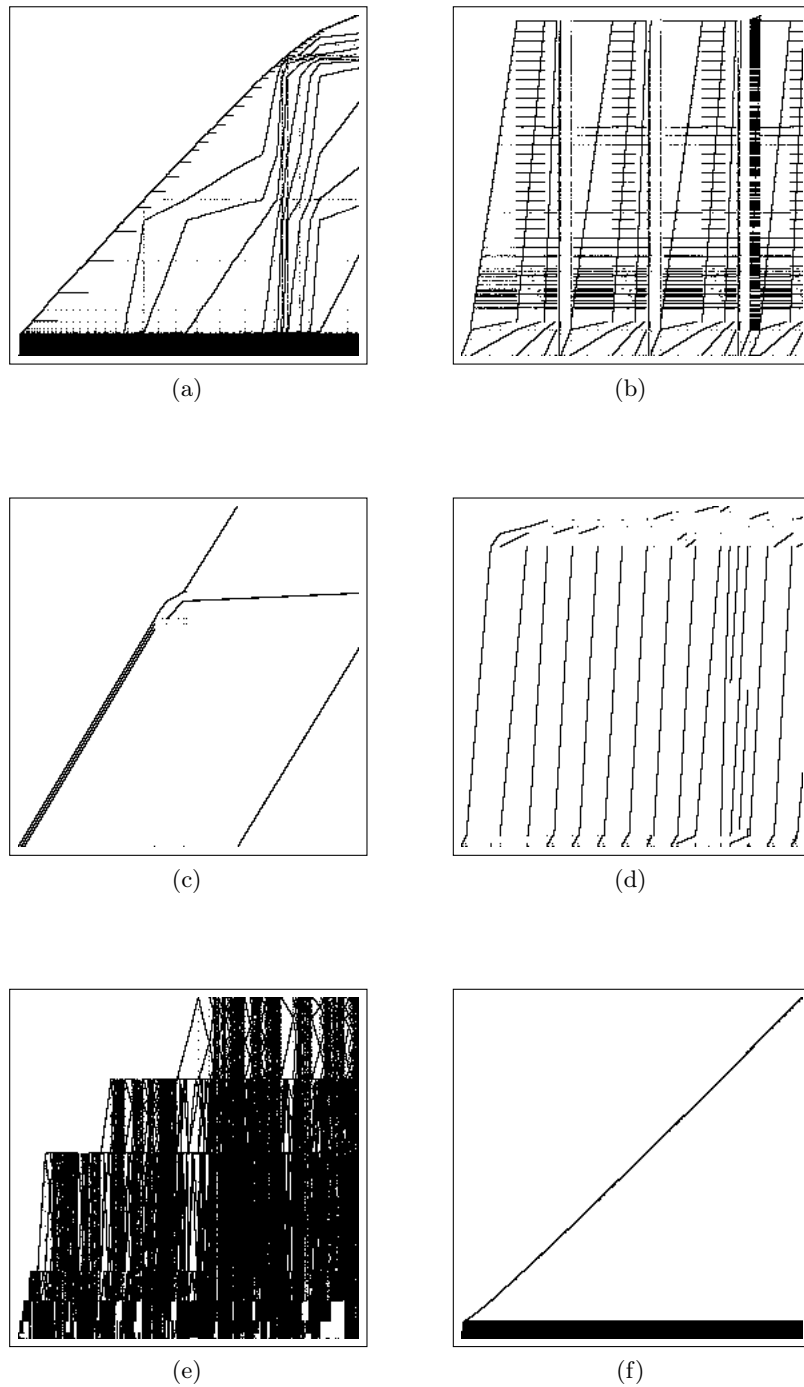


Figure 0.3: Γραφήματα με την χρήση `matplotlib` για (a) `bc-0` (b) `bfs-10` (c) `410.bwaves-s0` (d) `433.milc-s1` (e) `605.mcf-s8` (f) `cc-5 trace αρχεία`.

Κατά την εξερεύνηση των τεχνικών απεικόνισης εικόνων και του τρόπου απεικόνισης των πληροφοριών με μεγαλύτερη ακρίβεια, βρήκαμε ορισμένες αρκετά ενδιαφέρουσες πτυχές. Σε πολλά από τα σημεία αναφοράς, όπως αναμενόταν, λόγω της μεγάλης "συμπύκνωσης" που έχει γίνει για να "συμπυκνωθούν" όλα τα δεδομένα τους σε μια εικόνα μερικών εκατοντάδων εικονοστοιχείων (pixel) σε κάθε άξονα, χάνεται τεράστιος όγκος πολύτιμης πληροφορίας. Συνεχίζουμε με την εστίαση (zoom-in) σε συγκεκριμένα τμήματα αυτών των διαγραμμάτων. Μελετώντας τη συμπεριφορά διαφορετικών benchmarks βρήκαμε ορισμένες ενδιαφέρουσες περιπτώσεις. Όταν υπάρχει μεγάλος όγκος δεδομένων/σημείων που συμπυκνώνεται σε ένα

συγκεκριμένο εύρος σελίδας (Δy), αυτό οδηγεί, όταν απεικονίζεται ολόκληρο το σύνολο δεδομένων, σε ένα μαύρο ορθογώνιο στην εικόνα. Περιγράφουμε αυτό το σενάριο ως "Black Bar" benchmarks, τέτοια benchmarks είναι τα:

- bc
- cc
- 605.mcf
- sssp
- pr

Στον Πίνακα (Table) 0.2 βλέπουμε το διάστημα των σελίδων που κάθε ένα από τα παραπάνω benchmark παρατηρούμε το φαινόμενο του Black Bar, μαζί με τα συνολικές αστοχίες (misses) και το ποσοστό αυτών επί του συνόλου των αστοχιών του benchmark.

<i>Black Bar</i> benchmarks				
Benchmark	Αρχική σελίδα: y_t	Τελική Σελίδα: y'_t	# Αστοχιών	Ποσοστό αστοχιών επί του συνόλου (%)
bc	0	2200	11.75M	89
sssp	0	1300	4.5M	70
cc	0	1200	6M	79
pr	3000	4200	14M	79
605.mcf	0	2200	8M	30

Table 0.2: Το διάστημα σελίδων που αντιστοιχεί το *Black Bar* κομμάτι για κάθε ένα από τα benchmarks, το σύνολο των αστοχιών εντός αυτού του διαστήματος σελίδων καθώς και το ποσοστό επί του συνόλου των αστοχιών

Παράδειγμα εστίασης (zoom-in) σε Black Bar κομμάτια φαίνονται στο Γράφημα (Figure) 0.4.

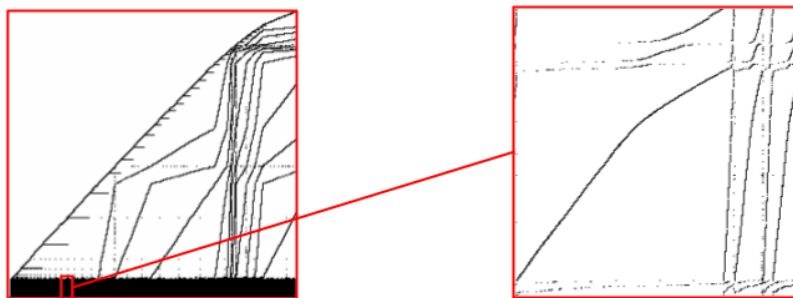


Figure 0.4: Παράδειγμα εστίασης σε περιοχή Black Bar στο bc-0 benchmark.

Για λόγους πολυπλοκότητας, δεν μπορούμε να έχουμε εικόνες με μέγεθος 2000 pixel στον άξονα y . Συνεπώς γίνεται χρήση εικόνων με μέγεθος άξονα y από 32 έως 128 pixel. Η απόφαση αυτή σημαίνει ότι κάθε εικονοστοιχείο στον άξονα y αντιστοιχεί σε ένα συγκεκριμένα εύρος τιμών σελίδων δy , όπου $\delta y = \Delta y / (\text{αριθμςεικονοστοιχεωνστονξονα}y)$. Για παράδειγμα, για το σημείο αναφοράς *sssp* έχουμε $\Delta y = 1300 - 0 = 1300$ το οποίο, για μια εικόνα με μέγεθος 128 εικονοστοιχειων στον άξονα y , μας δίνει $\delta y = \frac{1300}{128} \approx 10$ σελίδες/εικονοστοιχείο. Αυτό σημαίνει ότι κάθε εικονοστοιχείο στον άξονα y αντιστοιχεί σε περίπου 10 σελίδες. Η τιμή της θέσης εικονοστοιχείου όπου αντιστοιχεί μια σελίδα y_t στην τελική εικόνα θα συμβολίζεται με \bar{y}_t . Το Γράφημα (Figure) 0.5 απεικονίζει τη διαδικασία υπολογισμού της τιμής \bar{y}_t με ένα παράδειγμα και στην συνέχεια το Γράφημα (Figure) 0.6 απεικονίζει την διαδικασία μετατροπής της τελευταίας χρονοσειράς σε εικόνα.

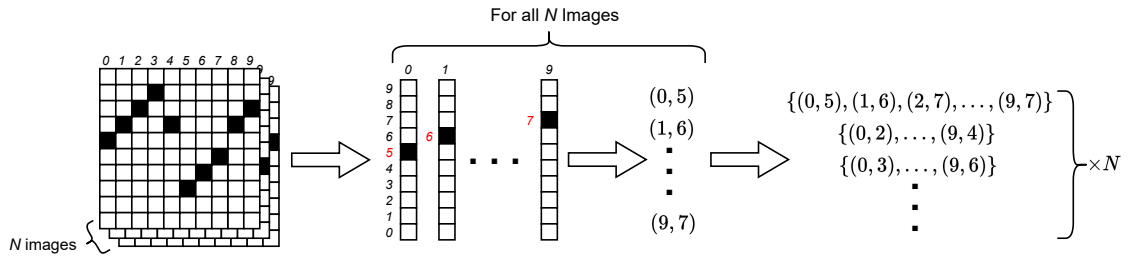


Figure 0.7: Μετατροπή εικόνας πίσω σε χρονοσειρά.

Value: $y_t = 254$

$$\delta y = \frac{\Delta y}{\# \text{ y-axis pixels}} = \frac{1000}{10} = 100$$

$$\bar{y}_t = \lfloor \frac{y_t}{\delta y} \rfloor = \lfloor \frac{254}{100} \rfloor = \lfloor 2.54 \rfloor = 2$$

$\delta y = 100 \text{ pages}$
 $\Delta y = 1000 \text{ pages}$

Figure 0.5: Παράδειγμα μετατροπής μιας τιμής της χρονοσειράς σε εικόνα μεγέθους 1x10.

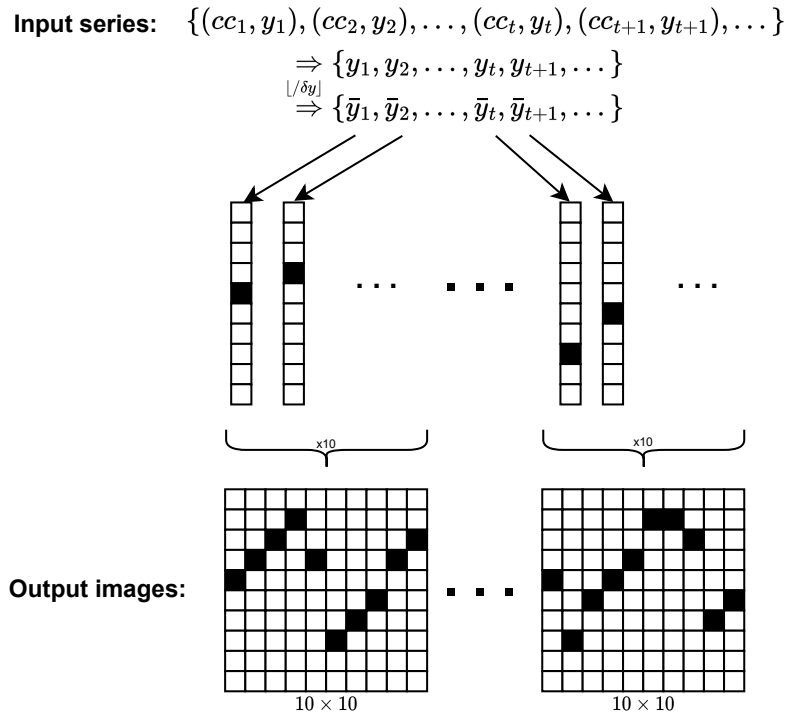


Figure 0.6: Απλοποιημένο διάγραμμα που απεικονίζει την διαδικασία οπτικοποίησης της χρονοσειράς εισόδου σε εικόνες μεγέθους 10x10.

Η διαδικασία της επιστροφής απο εικόνα σε χρονοσειρά είναι παρόμοια με αυτή της δημιουργίας της εικόνας. Ένα γράφημα που οπτικοποιεί την παραπάνω διαδικασία φαίνεται στο (Figure) 0.7

Συνεπώς οι εικόνες που δημιουργήσαμε θα έχουν τα εξείς χαρακτηριστικά:

- Μόνο τα benchmarks bc, sssp, cc, pr και 605.mcf.

- Μόνο σημεία όπου η τιμή της σελίδας βρίσκεται εντός των εύρων που δίνονται στον Πίνακα(Table) 0.2.
- Αναπαράσταση κάθε σημείου, ένα προς ένα, σε ένα συγκεκριμένο εικονοστοιχείο στον άξονα x, δίνοντάς μας δεκάδες χιλιάδες εικόνες ανά benchmark.
- Μεγέθη σε κάθε άξονα που κυμαίνονται από 64 έως 512 εικονοστοιχεία.
- Δυαδικές εικόνες (εικόνες με τιμές 0 ή 1 σε κάθε εικονοστοιχείο).

Τέλος για την δημιουργία του συνόλου εικόνας εισόδου καθώς και την αναμενόμενη πρόβλεψη σε κάθε περίπτωση δείχνουμε ένα παράδειγμα στο Γράφημα (Figure) 0.8. Όπου η εικόνα εισόδου και η εικόνα πρόβλεψη έχουν ίδιο μέγεθος στον άξονα y αλλά μπορούν να έχουν άλλο μέγεθος όσο αφορά τον άξονα x. Πιο συγκεκριμένα, το μέγεθος του άξονα x στην εικόνα εισόδου, αντιστοιχεί στην ιστορία πρόβλεψης (forecasting history, f_{hist}), δηλαδή το πλήθος παρελθοντικών τιμών που θα δοθούν ως πληροφορία στο μοντέλο για να κάνει την πρόβλεψη. Το μέγεθος στον άξονα x στην εικόνα πρόβλεψη αντιστοιχεί στον ορίζοντα πρόβλεψης (forecasting horizon, f_{hrz} ή f_h) δηλαδή πόσο "μακριά" στο μέλλον θα προβλέψει το μοντέλο.

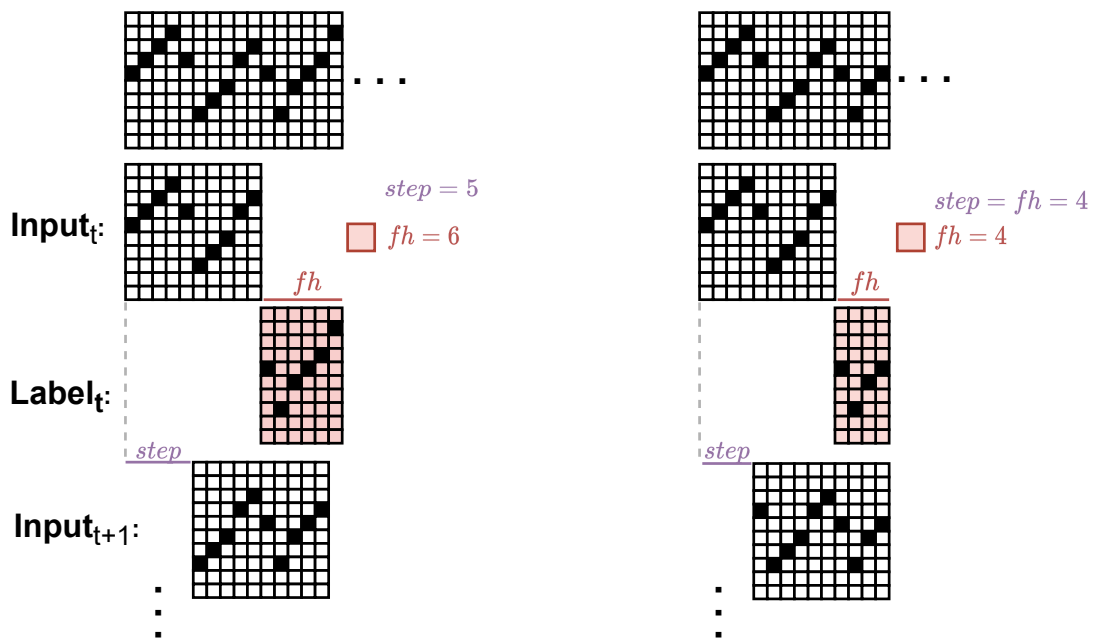


Figure 0.8: Είσοδος και αναμενόμενη πρόβλεψη για $f_h=6$ and $f_h=4$

0.5 Πρόβλεψη με Μεθόδους Μηχανικής Μάθησης Βασισμένες σε Εικόνες

Στο υποκεφάλαιο αυτό παρουσιάζουμε το τελικό μοντέλο που χρησιμοποιήθηκε για την πρόβλεψη μελλοντικών προσβάσεων στην μνήμη. Αρχικά εξετάσαμε προεκπαιδευμένα Συνελικτικά Νευρωνικά Δίκτυα (CNN) για το πρόβλημα μας, τα οποία είναι χρήσιμα σε διάφορες εργασίες. Ωστόσο, η αποτελεσματικότητά τους περιορίζεται στην περίπτωση μας, όπου οι εικόνες έχουν μια αρκετά συγκεκριμένη δομή (δυαδικές εικόνες, μια τιμή ανά στήλη κλπ.). Τα μοντέλα αυτά εκπαιδεύονται σε μεγάλα σύνολα δεδομένων, κυρίως εικόνες του πραγματικού κόσμου, και μαθαίνουν να αναγνωρίζουν συγκεκριμένα μοτίβα και χαρακτηριστικά. Όμως, όταν έρχονται αντιμέτωπα με τις δικές μας εικόνες, οι οποίες έχουν αρκετά διαφορετική δομή, δυσκολεύονται. Δοκιμάστηκαν επιπλέον μοντέλα πρόβλεψης επόμενου καρέ (next-frame prediction) το οποία

πάλι φάνηκαν να μην είναι κατάλληλα για εμάς. Ομοίως με πριν, έχουν σχεδιαστεί για εικόνες και βίντεο του πραγματικού κόσμου, ενώ τα δικά μας δεδομένα είναι πολύ διαφορετικά. Για να επιτύχουμε βέλτιστη απόδοση, χρειαζόμαστε μοντέλα που θα εκπαιδευτούν και θα σχεδιαστούν για το δικό μας σύνολο δεδομένων. Μόνο έτσι θα μπορούν να προσαρμοστούν στις ιδιαιτερότητες των εικόνων μας και να μας δώσουν τα επιθυμητά αποτελέσματα.

Το μοντέλο που καταλήξαμε, το οποίο ονομάσαμε ως *ConvLSTM-One-Img* έχει επιρροές από τα προβλήματα Next-Frame prediction καθώς και από τα Autoencoder CNN . Αποτελείται δηλαδή από ένα κομμάτι μείωσης διαστάσεων (down-sampling), όπου γίνεται χρήση ConvLSTM επιπέδων. Στην συνέχεια εισάγετε ένα κρυφό επίπεδο (dense layer) και μετά ένα κομμάτι για την επιστροφή στην αρχική διάσταση (up-sampling), με την χρήση ConvLSTM επιπέδων ξανά. Με τον παραπάνω τρόπο, χρησιμοποιώντας τον κατάλληλο αριθμό παραμέτρων στο κρυφό ενδιάμεσο επίπεδο, μπορούμε να προσαρμόσουμε το μέγεθος της εικόνας εξόδου στην τιμή που θέλουμε. Η λεπτομερής αρχιτεκτονική του μοντέλου που αναπτύχθηκε στο TensorFlow μπορεί να εξεταστεί στο Γράφημα (Figure)0.9.

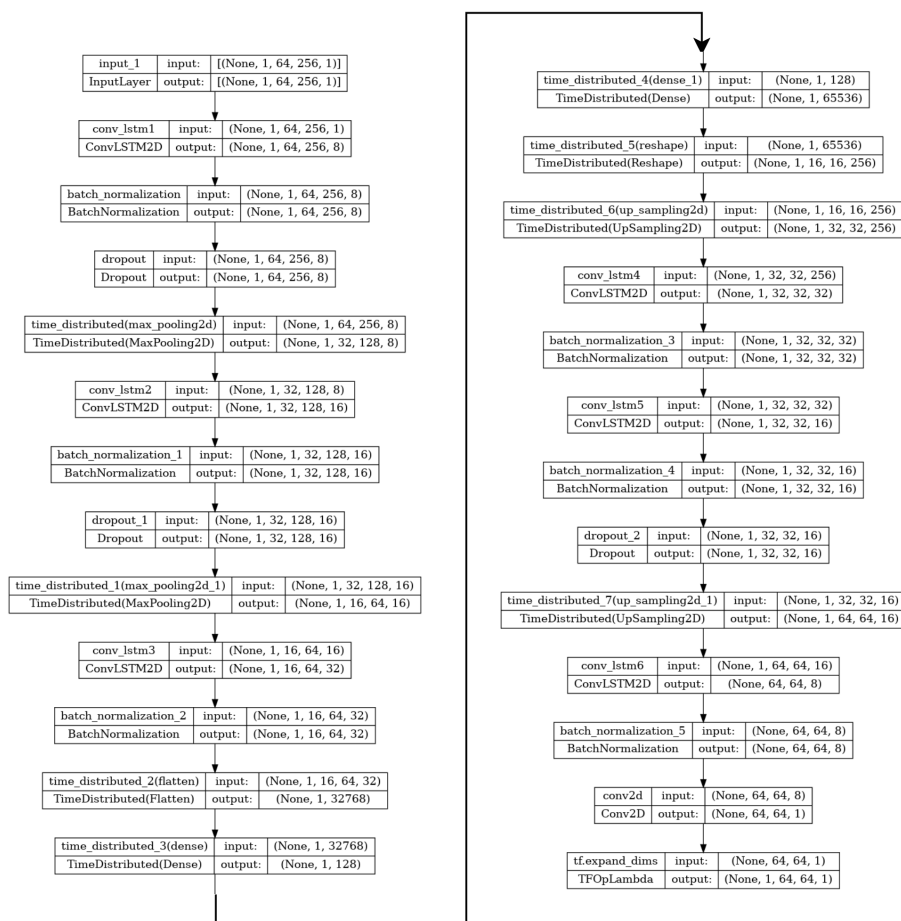


Figure 0.9: Αρχιτεκτονική Μοντέλου ConvLSTM-One-Img. Οι τιμές και οι διαστάσεις που εμφανίζονται προέρχονται από ένα τυχαίο παράδειγμα, καθώς εξαρτώνται από το μέγεθος της εικόνας εισόδου.

Δυστυχώς, λόγω του μεγάλου αριθμού μοντέλων που χρειάστηκε να εκπαιδευσουμε και να πειραματιστούμε, η χρήση μεθόδων όπως blocked cross-validation, όπου γίνεται πολλαπλές φορές εκπαίδευση σε κομμάτι του dataset και έλεγχος (testing) στο αμέσως επόμενο κομμάτι, θα ήταν εξαιρετικά χρονοβόρα. Σε όλο την υπόλοιπη εργασία, τα αποτελέσματα βασίζονται

στην ιδέα (όπως εφαρμόζεται και στα [57, 77]) ότι εκπαιδεύουμε το μοντέλο στο πρώτο κομμάτι του συνόλου δεδομένων, συγκεκριμένα στο 10-30 %, και στη συνέχεια το ελέγχουμε (test) στο υπόλοιπο τμήμα. Το πρώτο 10% παραλείπεται επειδή ([77]) παρατηρήσαμε ότι σε αυτό το τμήμα του συνόλου δεδομένων τα δεδομένα είναι πολύ πιο "αραιά" και με ιδιαίτερη συμπεριφορά, οδηγώντας τελικά σε χειρότερα αποτελέσματα από ό,τι όταν συμπεριλαμβάνεται. Λαμβάνοντας υπόψη τα παραπάνω, είναι προφανές ότι δημιουργούμε ένα δύσκολο πρόβλημα προς επίλυση, επειδή, όπως ήδη αναφέρθηκε, το σύνολο δεδομένων φαίνεται να έχει διαφορετική συμπεριφορά σε διαφορετικά τμήματα του συνόλου δεδομένων. Επίσης, εκτός εάν αναφέρεται διαφορετικά, όλες οι δοκιμές έγιναν για το αρχείο trace bc-0.

Χρησιμοποιήθηκαν διάφορες μετρικές για την αξιολόγηση των αποτελεσμάτων, ωστόσο η βασική μετρική που αξιολογούμε εν τέλει τα αποτελέσματα, είναι αυτή του Accuracy. Δηλαδή το ποσοστό των εύστοχων προβλέψεων, προς το σύνολο των προβλέψεων. Άλλες μετρικές που χρησιμοποιήθηκαν είναι η Intersection over Union (IoU), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) και Mean Absolute Percentage Error (MAPE). Όλες αυτές οι μετρικές μας δείχνουν πόσο "κοντά" είναι η πρόβλεψή μας, από τις πραγματικές τιμές, ωστόσο, μόνο η μετρική του Accuracy δίνει την πραγματική απόδοση των δοκιμών μας.

Στην συνέχεια προχωρήσαμε σε δοκιμές για διαφορετικές τιμές για την ιστορία πρόβλεψης (forecasting history), μέγεθος το άξονα y και μήκος πρόβλεψης (forecasting horizon). Στο σημείο αυτό θα εισάγουμε τα συμπεράσματα από τις παραπάνω δοκιμάσεις. Γραφικές παραστάσεις με τα αποτελέσματα θα δοθούν στο επόμενο υποκεφάλαιο.

Για την επίδοση του μοντέλου σε διαφορετικές τιμές ιστορικού πρόβλεψης (forecasting history). Το ConvLSTM-One-Img παρουσίασε την καλύτερη επίδοση σε τιμή ιστορικού πρόβλεψης 128 ή 256. Σε περαιτέρω δοκιμές θα χρησιμοποιείται η τιμή των 128 τιμών για ιστορικό. Στη συνέχεια, διερευνήθηκαν διαφορετικές τιμές για τον άξονα y . Η τιμή του άξονα y καθορίζει πόση συμπίεση πραγματοποιείται, επιλέχθηκε η τιμή 64 για ισορροπία μεταξύ της συμπίεσης και της ακρίβειας.

Τέλος, αξιολογήσαμε την επίδοση για διαφορετικές τιμές του ορίζοντα πρόβλεψης (forecasting horizon). Παρόμοια με τον άξονα y , πρέπει να αποδεχθούμε έναν συμβιβασμό μεταξύ του μήκους του ορίζοντα πρόβλεψης και της ακρίβειας. Συνεχίζουμε με τιμές ορίζοντα πρόβλεψης από 16 έως 64 λόγω της υψηλής τους απόδοσης και του επαρκούς μήκους πρόβλεψης.

0.6 Σύγκριση με Άλλες Τεχνικές Πρόβλεψης Χρονοσειρών

Στο προηγούμενο υποκεφάλαιο, προτάθηκε μια εναλλακτική μέθοδος για την πρόβλεψη μελλοντικών σελίδων. Αυτή η λύση αξιοποιεί τις δυνατότητες της όρασης υπολογιστών και της βαθιάς μηχανικής μάθησης για να μάθει μοτίβα στα δεδομένα και να προβλέψει μελλοντικές σελίδες. Σε αυτό το κεφάλαιο, θα συγκρίνουμε το μοντέλο της προσέγγισης που βασίζεται σε εικόνες, το ConvLSTM-One-Img, με την πιο κοινή προσέγγιση βαθιάς μάθησης στον τομέα της ανάλυσης χρονοσειρών και της πρόβλεψης, το LSTM.

Αρχικά κάναμε μια γρήγορη σύγκριση με στατιστικές μεθόδους πρόβλεψης χρονοσειρών. Πιο συγκεκριμένα έγινε δοκιμή του Exponential Smoothing και το Autoregressive Integrated Moving Average (ARIMA) Οι δύο προηγούμενοι μέθοδοι παρουσίασαν χαμηλότερο Accuracy για όλες του ορίζοντες πρόβλεψης που εξετάστηκαν, συγκριτικά με την προσέγγισή μας που βασίζεται σε εικόνες, με διαφορά Accuracy στο 10-25%. Η επίδοση επιδεινώνεται όσο μεγαλύτερος γίνεται ο ορίζοντας πρόβλεψης. Μια ενδιαφέρουσα παρατήρηση είναι ότι οι μετρήσεις επιδόσεων που βασίζονται σε σφάλματα (RMSE, MAE, MAPE) δεν είναι αναλογικά όσο κακές όσο οι μετρήσεις Accuracy ή IoU. Αυτό εξηγείται από το γεγονός ότι και τα δύο μοντέλα, όπως αποδεικνύεται εξετάζοντας τις εικόνες πρόβλεψης που συμπεριλήφθηκαν 0.10,

προβλέπουν μια μέση τιμή από το ιστορικό δεδομένων τους. Έτσι, ειδικά σε προβλέψεις με μικρό ορίζοντα πρόβλεψης, μπορεί να δώσουν λανθασμένη πρόβλεψη (χαμηλό Accuracy και IoU) αλλά η διαφορά μεταξύ της πρόβλεψης και της πραγματικής τιμής είναι συνήθως αρκετά μικρή, με αποτέλεσμα οι μετρήσεις σφάλματος να μην είναι τόσο κακές όσο θα περίμενε κανείς. Δεν συνεχίστηκαν περαιτέρω δοκιμές και πειράματα με αυτά τα μοντέλα, καθώς ούτε επέδειξαν ανταγωνιστική επίδοση ούτε υπάρχει περαιτέρω ενδιαφέρον για εις βάθος εξερεύνηση τους.



(a) *Exponential Smoothing*



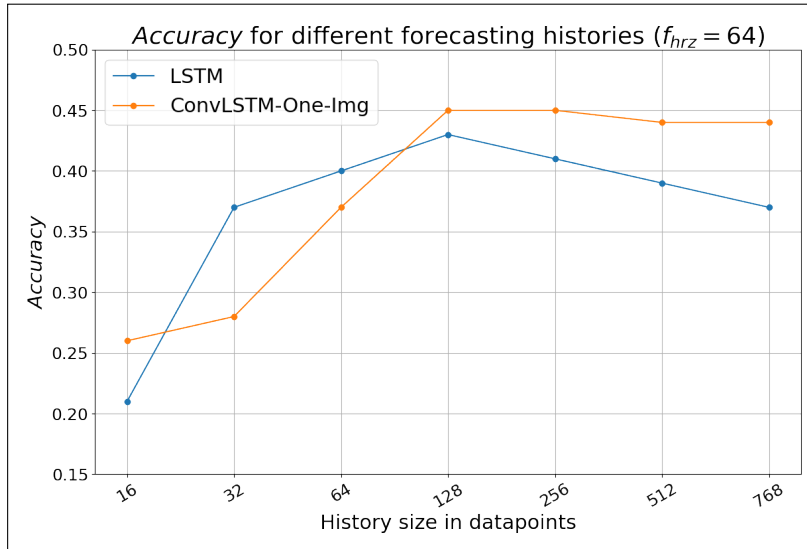
(b) *ARIMA*



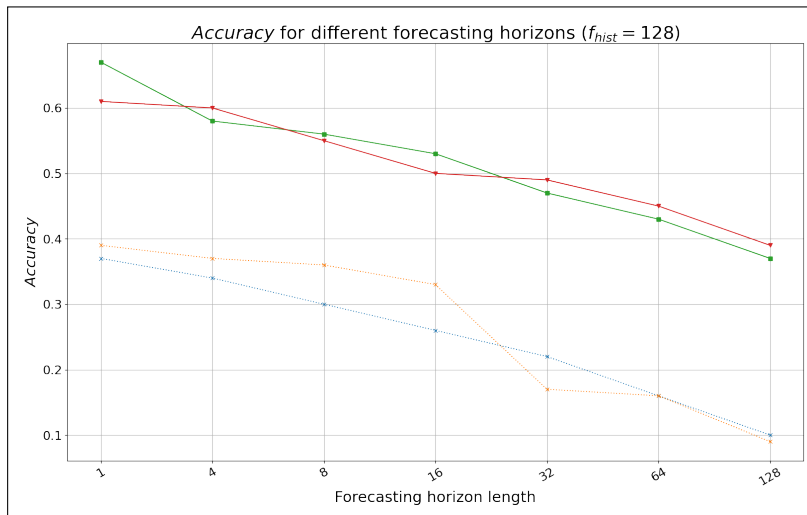
(c) *GroundTruth*

Figure 0.10: Παράδειγμα απεικόνισης προβλέψεων με Exponential Smoothing και ARIMA μαζί με την πραγματικές τιμές, για ορίζοντα πρόβλεψης $fh = 32$. Σύνολο 15 προβλέψεων έχουν συνενωθεί για κάθε μία από τις εικόνες.

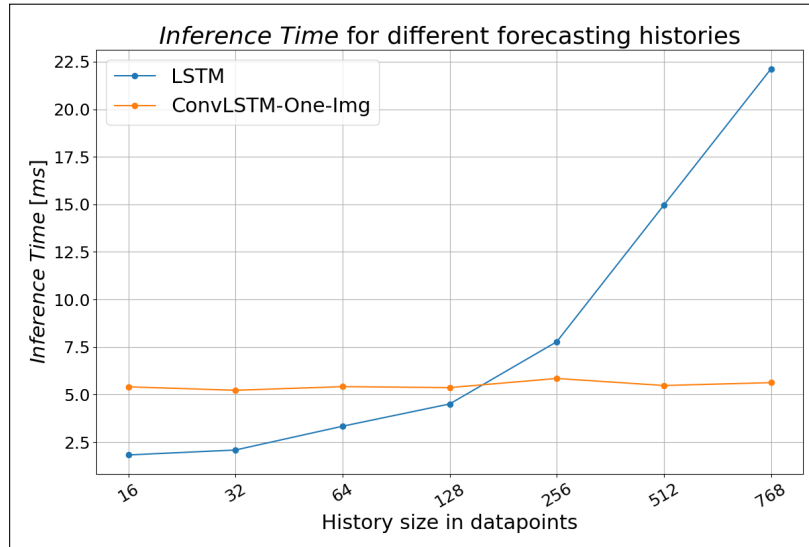
Στο σημείο θα εξετάσουμε την χρήση LSTM μοντέλων, που είναι η πιο σύνηθες προσέγγιση στην πρόβλεψη χρονοσειρών. Το μοντέλο που χρησιμοποιήθηκε έχει ως είσοδο μια μονοδιάστατη χρονοσειρά (μήκους ίσο με την ιστορία πρόβλεψης που επιθυμούμε) η οποία περνάει από τρία LSTM στρώματα και στη συνέχεια από δύο dense στρώματα, με το τελευταίο να παρέχει την έξοδο, η οποία έχει μήκος που αντιστοιχεί στον ορίζοντα πρόβλεψης. Στην συνέχεια προχωρήσαμε σε δοκιμές για διαφορετικές τιμές για την ιστορία πρόβλεψης (forecasting history) και στο μήκος πρόβλεψης (forecasting horizon) όπως έγινε και προηγουμένως με το μοντέλο *ConvLSTM-One-Img* καθώς και τις μετρικές του Inference Time (χρόνου που χρειάζεται το μοντέλο για την πρόβλεψη). Παραθέτουμε γραφικές παραστάσεις για την μετρική του Accuracy και του Inference Time στο Γράφημα(Figure) 0.11, όπου απεικονίζεται το *ConvLSTM-One-Img* και το *LSTM* μοντέλο.



(a)



(b)



(c)

Figure 0.11: Σύγκριση *ConvLSTM-One-Img* και *LSTM* μοντέλων. (a) Σύγκριση των τιμών Accuracy για διαφορετικές τιμές ιστορίας πρόβλεψης, (b) Σύγκριση των τιμών Accuracy για διαφορετικές τιμές ορίζοντα πρόβλεψης (έχουν συμπεριληφθεί οι στατιστικές μέθοδοι Exponential Smoothing και ARIMA) και (c) Σύγκριση του Inference Time για διαφορετικές τιμές ιστορίας πρόβλεψης

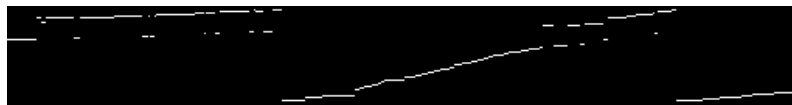
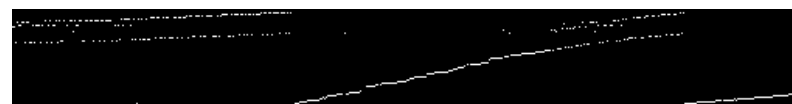
(a) *ConvLSTM-One-Img*(b) *LSTM*(c) *Ground Truth*

Figure 0.12: Παράδειγμα απεικόνισης προβλέψεων *ConvLSTM-One-Img* ($Accuracy=0.36$) και *LSTM* ($Accuracy=0.30$) και των πραγματικών τιμών, για ορίζοντα πρόβλεψης $fh = 32$ και ιστορία πρόβλεψης 128. Σύνολο 16 προβλέψεων έχουν συνενωθεί για κάθε μία από τις εικόνες

Παρατηρώντας το Γράφημα(Figure) 0.11(a) η αύξηση του ιστορικού πρόβλεψης βελτιώνει την επίδοση και των δύο μοντέλων *LSTM* και *ConvLSTM-One-Img*, και τα 2 μοντέλα δίνουν την καλύτερη τιμή Accuracy για τιμή ίση με 128. Το *LSTM* υπερτερεί σε μικρότερα ιστορικά

πρόβλεψης, όπου αρχίζει να “προβλέπει” καλύτερα με σημαντικά μικρότερο forecasting history σε σχέση με το *ConvLSTM-One-Img*. Από την άλλη, το *ConvLSTM-One-Img* υπερέρχει σε μεγαλύτερα ιστορικά (128 και πάνω).

Σημαντικό είναι επίσης, πως το *ConvLSTM-One-Img* έχει σταθερό χρόνο εκτέλεσης ανεξάρτητα του ιστορικού, ενώ ο χρόνος εκτέλεσης του *LSTM* αυξάνεται σημαντικά με μεγαλύτερα ιστορικά (0.11(c)). Για ιστορικά μέχρι 128 ωστόσο, το *LSTM* έχει ταχύτερη εκτέλεση.

Στο Γράφημα(Figure) 0.11(b), εξερευνούμε πώς συγκρίνονται τα μοντέλα *LSTM* και *ConvLSTM-One-Img* για διαφορετικές τιμές μήκους πρόβλεψης (forecasting horizon). Συμπεριλάβαμε τις προαναφερθείσες στατιστικές μεθόδους. Τα τελευταία μοντέλα αποδίδουν σημαντικά χειρότερα από τα μοντέλα μηχανικής μάθησης που εφαρμόστηκαν και απορρίπτονται όπως αναφέραμε νωρίτερα. Τα μοντέλα *LSTM* και *ConvLSTM-One-Img* έχουν εξαιρετικά συγκρίσιμα αποτελέσματα και ακολουθούν σχεδόν πανομοιότυπη συμπεριφορά για όλες τις δοκιμασμένες τιμές. Μια πιθανή εξήγηση για τα παρόμοια αποτελέσματα μπορεί να είναι η επιλεγμένη απεικόνιση, η οποία να είναι αυτή που περιορίζει την δυνατότητα των μοντέλων μας να προβλέψουν καλύτερα και όχι τα μοντέλα καθαυτά.

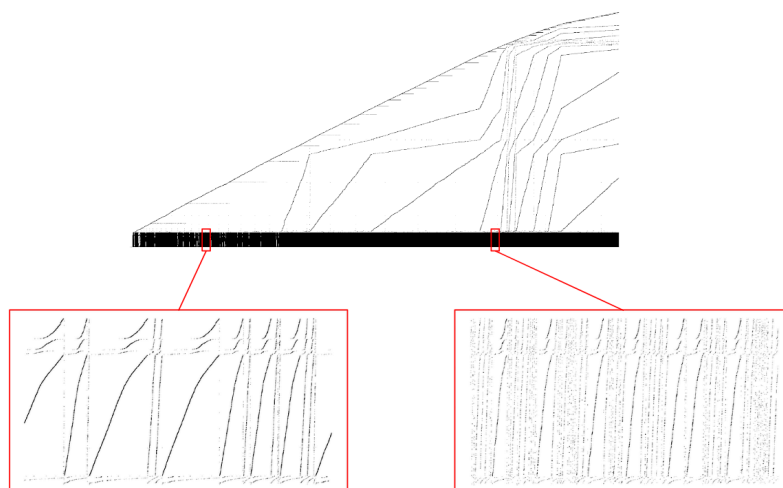


Figure 0.13: Οπτική απεικόνιση της διαφοράς στην "πυκνότητα" των μοτίβων (strides) σε διαφορετικά σημεία στο ίδιο benchmark. Τα 2 κόκκινα εισασμένα (zoomed-in) παράθυρα είναι ίδιου μεγέθους Δcc

Όπως αναφέρθηκε προηγουμένως, όλα τα μοντέλα εκπαιδεύτηκαν σε 10 – 30% του benchmark *bc* και στη συνέχεια δοκιμάστηκαν στο υπόλοιπο. Αυτό φυσικά οδηγεί σε χειρότερα αποτελέσματα επειδή, όπως φαίνεται στο Σχήμα 0.13, η συμπεριφορά του benchmark αλλάζει κατά τη διάρκεια της εκτέλεσής του, οδηγώντας στη χρήση ενός μοντέλου σε μοτίβα δεδομένων που δεν έχουν παρατηρηθεί. Για να δούμε πώς αποδίδουν τα μοντέλα μας καθώς προχωράμε στο benchmark, δοκιμάσαμε για κάθε 10% ανεξάρτητα. Δοκιμάσαμε με μοντέλα *LSTM* και *ConvLSTM-One-Img* με ιστορικό εισόδου 128 και ορίζοντα πρόβλεψης 32. Τα αποτελέσματα φαίνονται στο Γράφημα (Figure) 0.14.

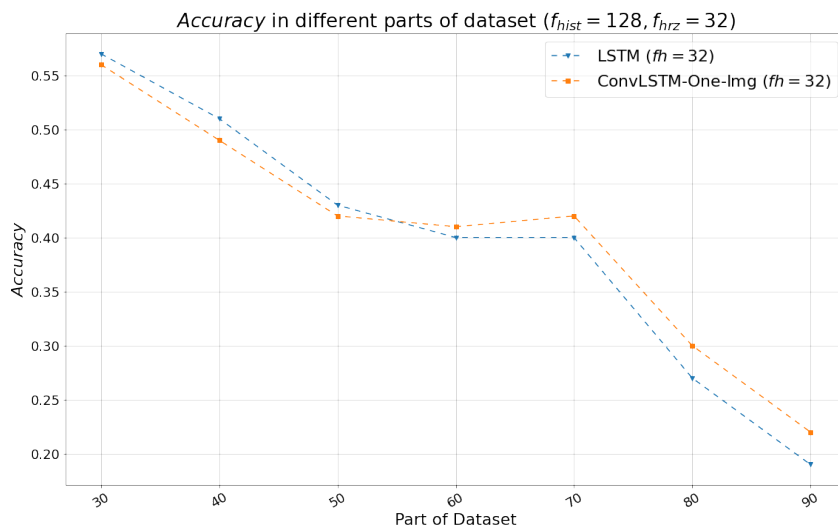


Figure 0.14: Σύγκριση *ConvLSTM-One-Img* και *LSTM* μοντέλων για εκπαίδευση στο 10-30% του dataset και έλεγχο ανα 10% για το υπόλοιπο dataset.

Εξετάζοντας τις τιμές απο το Γράφημα (Figure) 0.14 διαπιστώσαμε ότι στο αρχικό κομμάτι του dataset, αμέσως μετά το σύνολο εκπαίδευσης, το *LSTM* μοντέλο παρουσίασε ελαφρώς καλύτερες επιδόσεις. Καθώς προχωράμε στο σύνολο δεδομένων, όπου οι πληροφορίες αρχίζουν να έχουν διαφορετικά μοτίβα και συμπεριφορά, η επίδοση και των δύο μοντέλων (*LSTM* και *ConvLSTM-One-Img*) είναι χειρότερη. Ωστόσο, το μοντέλο *ConvLSTM-One-Img* έχει ελαφρώς καλύτερο Accuracy σε αυτά τα τελευταία τμήματα του συνόλου δεδομένων, υποδηλώνοντας μια ισχυρότερη ικανότητα γενίκευσης σε σύγκριση με το *LSTM* μοντέλο.

0.7 Συμπεράσματα και Μελλοντικές Προεκτάσεις

Η αυξανόμενη διαφορά μεταξύ ισχύος επεξεργασίας και ταχύτητας μνήμης δημιουργεί προβλήματα επιδόσεων. Η πρόβλεψη μελλοντικών σελίδων μνήμης μπορεί να βελτιώσει υπάρχουσες μεθόδους διαχείρισης δεδομένων. Η διπλωματική αυτή παρουσιάζει μια νέα προσέγγιση για την πρόβλεψη, παραπάνω του ενός, μελλοντικών σελίδων μνήμης, αξιοποιώντας μεθόδους όρασης υπολογιστών και μηχανικής μάθησης. Παίρνοντας έμπνευση από την πρόβλεψη χρηματοοικονομικών δεδομένων, η εργασία εξετάζει τη χρήση εικόνων και μεθόδων όρασης υπολογιστών για την πρόβλεψη μελλοντικών σελίδων που απαιτούνται από μια εφαρμογή. Αναλύθηκαν αρχεία πρόσβασης μνήμης και προτάθηκε μια οπτική απεικόνιση δεδομένων. Αυτή η απεικόνιση δίνεται για εκπαίδευση και αξιολόγηση σε μοντέλα μηχανικής μάθησης για την πρόβλεψη μελλοντικών προσβάσεων. Η σύγκριση με υπάρχουσες μεθόδους έδειξε ότι το μοντέλο που προτείνουμε ξεπερνά τις στατιστικές μεθόδους και είναι συγκρίσιμο με τις τρέχουσες καλύτερες λύσεις στην πρόβλεψη χρονοσειρών, τα *LSTM*. Σε ορισμένα σενάρια μάλιστα υπερτερεί του *LSTM* μοντέλο. Η διπλωματική θέτει τις βάσεις για την περαιτέρω ανάπτυξη τεχνικών μηχανικής μάθησης (Machine Learning, ML) με βάση τις εικόνες για την πρόβλεψη σελίδων, που μπορούν να οδηγήσουν σε σημαντικές βελτιώσεις των επιδόσεων των συστημάτων που περιορίζονται απο την ταχύτητα της μνήμης.

Δεδομένου πόσο αχαρτογράφητος ήταν αυτός ο τομέας έρευνας, προφανώς παρουσιάστηκαν αρκετες δυνατότητες για μελλοντική εργασία. Αρχικά με το τρομερή προοδο και ταχύτητα του machine learning θα εξερευνήσουμε τα Vision Transformer (ViT) [21], μορφή των οποίων χρησιμοποιήθηκε στο Drishyam [57]. Τα transformers ερχόμενα απο τα Large Language Models έχουν πολύ καλή συμπεριφορά όσο αφορά την ιστορία των δεδομένων, κάτι το οποίο

θα μπορούσε να αποδειχθεί πολύ χρήσιμο στο πρόβλημα μας.

Στην συνέχεια αυτο που θα ήθελε ιδιαίτερη περεταίρω εξερεύνηση είναι το visualization. Ομολογουμένως αυτή που επιλέχθηκε ήταν ιδιαίτερα απλή με αρκετά υποθέσεις, όπως η έμφαση στις Black-Bar περιοχές, την συμπίεση των σελίδων σε 64 pixel κλπ. Σε μια νέα προσέγγιση θα δοκιμάζαμε τεχνικές όπως την οπτικοποίηση των delta μεταξύ των σελίδων (δηλαδή την διαφορά μεταξύ σελίδων και όχι την τιμή των σελίδων) και την ένταξη του Μετρητή Προγράμματος (Program Counter), τεχνικές η οποίες έχουν χρησιμοποιηθεί σε διάφορες δημοσιεύσεις που χρησιμοποιούν μηχανική μάθηση και όχι [33, 42, 63]. Επιπλέον η δημιουργία εικόνων πιο πλούσια σε πληροφορία όπως τα Gramian Angular Field (GAF) [37, 84] και Recurrence Plots (RP) [23] θα ήταν κάτι που θα είχε μεγάλο ενδιαφέρον. Με όλα τα παραπάνω στόχος θα ήταν να δώσουμε περισσότερη πληροφορία στο μοντέλο και ενδεχομένως να ξεπεράσουμε τους περιορισμούς που είδαμε στην απόδοση τους.

Ένα κρίσιμο επόμενο βήμα θα ήταν η ενσωμάτωση κάποιας προσομοίωσης μικροαρχιτεκτονικής για μια πιο ολοκληρωμένη αξιολόγηση (π.χ. CMP\$im [39], McPat [49] και SimpleScalar [8]). Αυτό θα κάνει δυνατή την άμεση σύγκριση με υπάρχουσες μεθόδους. Οι προσομοιωτές μικροαρχιτεκτονικής μπορούν να μοντελοποιήσουν την λειτουργία πραγματικών επεξεργαστών, επιτρέποντας τη σύγκριση του μοντέλου μας βασισμένο στην μηχανικής μάθησης με υπάρχουσες τεχνικές (μηχανικής μάθησης και μη). Αυτό θα προσφέρει πολύτιμη πληροφορία για τα σχετικά πλεονεκτήματα και μειονεκτήματα κάθε προσέγγισης. Επιπλέον, θα μπορούμε να λαμβάνουμε ακριβείς μετρήσεις επιδόσεων. Οι προσομοιωτές διευκολύνουν την ακριβή μέτρηση κρίσιμων μετρητών επιδόσεων, όπως οι εντολές που εκτελούνται ανά κύκλο ρολογιού επεξεργαστή (Instruction Per Cycle, IPC) και ο χρόνος εκτέλεσης (Execution Time). Αυτά τα στοιχεία αντικατοπτρίζουν άμεσα την επίδραση της προφόρτωσης δεδομένων στην κρυφή μνήμη στην επίδοση του επεξεργαστή. Η ανάλυση αυτών των μετρικών θα βοηθήσει να ποσοτικοποιήσουμε την αποτελεσματικότητα του προτεινόμενου μοντέλου μας.

Τέλος, στο Memmap [78], οι συγγραφείς προτείνουν τη χρήση τριών μικρών και γενικευμένων LSTM μοντέλων αντί για ένα εξειδικευμένο μοντέλο για κάθε τύπο δεδομένων. Αυτή η προσέγγιση αποσκοπεί σε καλύτερη κλιμακωσιμότητα. Αναλόγως, θέλουμε να διερευνήσουμε τη δυνατότητα ανάπτυξης ενός γενικευμένου μοντέλου βασισμένου σε Συνελικτικά Νευρωνικά Δίκτυα (CNN).

Chapter 1

Introduction

1.1 Motivation

Modern computing systems, from personal devices to powerful data centers, have demonstrated significant rise in processing power. This remarkable progress comes with increasing challenges that demand innovative solutions. The continuously expanding gap between processing power and memory access speed, also known as memory wall([88]), leads to more compute cycles being lost in an idle state while data is being retrieved from the memory. It has been observed that modern applications can spend in some cases almost half of their execution time in this stage [43, 33]. Data center servers face a similar growing challenge. As their processing power increases, they are demanding more memory to handle numerous applications simultaneously. Traditional DRAM, although fast, is not a scalable option as it is expensive, limited in size, and consumes power constantly. A solution has been found in Hybrid Memory Systems (HMS). These combine the speed of DRAM with the scalability and lower power consumption of Non-Volatile Memory (NVM); the trade-off being that NVM has slower read and write access times. To bridge this gap, these systems rely on page schedulers [85, 22, 44].

To overcome these limitations and take advantage of the full potential of the systems, computer architecture leverages prediction and heuristics to enhance performance. Page prediction is a key method in this approach. This technique predicts the memory pages that will be required by an application in the near future. This enables and enhances proactive strategies like data prefetching (paired with an offset predictor). In this approach, the system pre-loads relevant data into the cache before it's truly needed, effectively eliminating the latency associated with fetching from DRAM [38, 9]. Furthermore, page predictors can inform page scheduling/migration/replacement algorithms for Hybrid memory systems in order to make better decisions when allocating pages. This optimization can significantly reduce memory latency [54, 22, 44]. In essence, page predictors pave the way for substantial performance improvements on both hardware and system levels.

Traditional data prefetchers rely on predicting future data needs based on past usage stored in tables. The table size scales with the program's active data, making them unsuitable for the massive and complex workloads of modern data centers [58, 33]. As data volume grows, the accuracy of the predictions significantly deteriorates. Current page scheduling and migrating algorithms for Hybrid memory systems primarily employ simple techniques like Least Recently Used (LRU) , Not Recently Used (NRU), and Translation Lookaside Buffer (TLB), highlighting the need for more sophisticated solutions to handle the evolving landscape of data processing.

Machine learning is gaining significant traction in these fields. Its evolving capabilities enable robust and accurate predictions, particularly for irregular memory access patterns. Machine learning can also offer longer forecasting horizons [71, 61], providing more information about upcoming pages and most likely leading to better performance. Multiple proposed ideas have explored the use of different machine learning algorithms in order to enhance the performance of the mentioned systems. These include the use of Recurrent Neural Network (RNNs) [78, 75, 44], Natural Language Processing (NLP) methods [33], Transformers [90] and Reinforcement Learning [22, 6]. We must note that most of the proposed methods, especially on hardware level data prefetching, even though they provide impressive performances, are still not feasible for practical implementations due to limitations in storage and computation speed. They are explored in order to make a case that machine learning can be used for solving such problems successfully and are constantly improving upon their feasibility.

Memory page prediction resembles a time series forecasting task. Past accessed pages act as a history, and are used to predict which page will be needed next. Traditional methods in timeseries forecasting include linear regression and exponential smoothing and ARIMA [80] whereas more recently there are a lot more machine learning solutions utilizing RNNs [47, 26], Transformers [24, 76, 87] and CNNs [47] etc. Recent work in forecasting time series of financial data [19], explored further the use of image-based machine learning methods in order to capture more complex information within the data. This approach involves transforming time series data into images, allowing models to exploit two-dimensional representations. This captures both temporal (time-based) and spatial (pattern-based) relationships within the data, which traditional numerical methods often miss. This capability leads to the development of more robust forecasting models

Inspired by the work in forecasting financial time series data using an image-based prediction model. This thesis explores the effectiveness of visualizing the page sequence timeseries information as an image and using image-based machine learning models, in order to improve page prediction performance by leveraging the spatial and temporal correlations of the data. The goal is to learn the memory access patterns and successfully predict future ones. Comparison will be done with traditional timeseries forecasting methods and with the currently most used approach of LSTM models. Coincidentally, very recent work that was proposed while this research was conducted, validates the approach taken by proposing a state-of-the-art image based solution that outperforms previous non-image machine learning solutions on the problem of prefetching a future address based on some given history [57]. Furthermore most related work, focuses on predicting one page or address into the future, while this work will focus on leveraging the extra potentials of machine learning and predicting longer forecasts.

This thesis aims to lay the foundations for supporting future use of computer vision, visualization and image-based pipelines for tasks such as page prediction; however it is not expected to propose a practical solution that can compete with the currently used hardware approaches.

1.2 Related Work

In this section, previous research work on the topics of interest is summarized. Previous publications could be grouped into traditional solutions and machine learning-based solutions.

1.2.1 Page Prediction

Page prediction can be found as a key component of modern data prefetchers on hardware level or for a page scheduler/migration/replacement algorithms on a system level.

Traditional/Non Machine Learning approaches

Exploring the data prefetchers further, traditional approaches do not implement page prediction and rather only address prediction. We will cover slightly the traditional and state-of-the-art non machine learning approaches. Then we will continue to the recent machine learning based methods where the need for page prediction will be introduced.

Modern processors mostly use two types of prefetchers:

(a) Stride prefetchers: These are simpler and more commonly implemented in modern processors. They watch for memory access patterns where the difference between addresses (delta) stays constant [42, 63, 64]. For example, if a program reads data at locations $addr_x$, $addr_x+10$, $addr_x+20$, and so on, the prefetcher predicts it will need $addr_x+30$ next and loads it in advance.

(b) Correlation prefetchers: These are more complex and less common as they are not as reliable as a single steady delta. Compared to stride prefetchers, they are better at forecasting more erratic patterns since they retain the historical data of memory accesses in big tables. Markov [41] and GHB prefetchers [60] are examples of correlation prefetchers. However, correlation prefetchers have a drawback: They require a lot of memory to store the access history, making them expensive and not ideal for use in modern multi-core processor systems.

Current state-of-the-art non machine learning solutions include:

Berti [59], which leverages information from the first-level (L1) cache to select the best local delta based on specific design criteria.

SPP [45] stands out as a state-of-the-art history-based prefetcher. It excels at predicting complex access patterns by leveraging a compressed history. Unlike other methods, SPP transcends page boundaries, maintaining prefetching even during pattern shifts across memory pages. Additionally, it dynamically adjusts its prefetching intensity based on the confidence level of its predictions.

Bingo [10], a proposed spatial data prefetcher, leverages short and long events to identify the optimal data pattern for prefetching. Notably, it maintains a storage-efficient design, requiring only a single history table to manage the link between access patterns and the corresponding events.

Finally, the Variable Length Delta Prefetcher (VLDP) [73] employs a more sophisticated approach. VLDP tracks the differences (deltas) between memory addresses accessed within the same physical page. This historical information is then used to predict the sequence of future memory accesses in new pages. However, unlike simpler methods, VLDP relies on numerous prediction tables to achieve this capability.

Machine Learning approaches

Turning towards the machine learning solutions, one of the most cited works can be found by

Hashemi et al. [33]. This research pioneers the use of machine learning for data prefetching. It establishes a connection between existing prefetching techniques and n-gram models employed in natural language processing and proceeds by demonstrating that recurrent neural networks could replace traditional methods eventually given the limitations in compute and storage at the time of writing.

Another RNN based approach can be found by Srivastava, Ajitesh et al. [78] called MemMAP. This research introduces small, efficient LSTM models capable of accurately predicting the next memory access. It shows that by retraining three compact models with a limited amount of data for most applications, they can achieve performance close to specialized, larger models. Progressing towards a more generalized solution and compressed solution, making it a more viable option for real world implementation.

More recent work which includes the use of Transformers is TransforMAP by Zhang, Pengmiao et al. [90]. This work leverages the powerful Transformer model to perform multiple cache line predictions learning directly from the whole addresses space. This approach avoids information loss and saves a token table in hardware. To the best of our knowledge this is one of the only work exploring memory prefetching multiple addresses.

The benefits of page forecasting

A major challenge, as explained in [75], when implementing machine learning solutions for data prefetching is the exponential growth of possible outcomes (class explosion). This renders predicting absolute memory addresses as individual classes impractical. However, we can mitigate this issue by decomposing the prediction task into two smaller problems:

- (a) OS page prediction: This stage focuses on anticipating the specific memory page the program will access next.

- (b) Offset prediction within a page: Once the target page is identified, this step predicts the precise location (offset) within that page where the data resides.

This highlights the potential of the proposed page predictor in this thesis for significantly improving data prefetching and was one of the motivating factors for this work.

Voyager, by Shi, Zhan et al. [75] is currently one of the best performing machine learning prefetchers with extensive work on optimizing its computation speed and size in order to become comparable with hardware prefetchers. In this very thorough work, a hierarchical structure that separates addresses into pages and offsets and that introduces a mechanism for learning important relations among pages and offsets is introduced. Being also the first work to prove that LSTM-based prefetchers can outperform existing hardware prefetchers improving upon metrics such as IPC and Coverage. By also reducing overhead in training cost, prediction latency this model represented a significant step towards a practical neural prefetcher.

Finally it is very important to include Drishyam by Mohapatra Shubdeep and Biswabandan Panda [57]. This novel prefetcher leverages computer vision techniques. It analyzes visual representations of memory access patterns to predict future with an image classification approach and image based SWIN Transformers [55]. Similar to Voyager [75] it breaks the problem in to two, one for the page accessed and another for the corresponding offset in that page. An embedding is created in each case and then a corresponding YBG spectrum image from that. The goal of the image creation is to predict the future page and offset separately. The results show that it provides high accuracy and handles a wider range of scenarios compared to Voyager, resulting in superior performance on a set of benchmarks known for their irregular access patterns. As stated in their work, their goal was to explore computer

vision in this field and not to propose a practical implementation. The research conducted in this thesis, which focused on incorporating images into the data prediction problem, received further validation after the publication of this work. While employing a distinct approach by framing it as a classification problem and predicting only one page/offset into the future, this subsequent study successfully demonstrated the potential benefits of using images in this context.

Page prediction for Hybrid Memory Systems

After exploring the related work on data prefetchers we will provide some work for Hybrid memory management algorithms too. Given the different nature of this problem, which focuses on either page migration, page replacement or page scheduling, we will be presenting only a couple of machine learning solutions that have been proposed and is directly relevant to our work and as an example of areas our work could contribute.

Kleio, by Doudali, Thaleia Dimitra et al. [22] is a hybrid page scheduler for hybrid memory solutions. More specifically, it combines traditional lightweight history-based data tiering method with intelligent placement decision based on RNNs. The later provides hybrid memory systems with fast and effective neural network training and prediction accuracy levels.

Recent work by Katsaragakis, Manolis et al. [44] proposes a novel way to manage hybrid DRAM/NVM storage leading to significantly improved performance and noteworthy savings in energy consumption. The proposed scheduler consists of three major components. (a) the Critical Page Selector, (b) the Adjacent LSTM-based Predictor (b) the History-based Predictor (similarly to [22]). The combination of the later three components provides the proposed page scheduling system.

Though neither solution necessitates a future page prediction (as in the proposed approach this thesis), a successful long-term page prediction capability could prove immensely valuable. This holds the potential to significantly enhance the performance of these systems.

1.3 Thesis Outline

This thesis presents a novel approach in learning memory access patterns and forecasting multiple future pages, by leveraging state-of-the-art computer vision and machine learning methods. 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. The remainder of this thesis document is organized as follows:

In Chapter 2, the different methods utilized across this thesis are explained and some further background information is given for better understanding of their use.

Chapter 3 introduces us to the datasets. Starting by describing the information about the source of the data, what type of information they contain and how is that information structured within the given raw data files. After the above are understood, we proceed with presenting how we leverage this data, process it, and eventually produce the final timeseries data we will be working with. The exact process of visualizing the aforementioned data to binary images is presented and then the methodology to decompose the images back to numeric data, that will be used in order to compare with ground truth.

Chapter 4 explores the image-based machine learning models. More specifically after

introducing the reader to the field of machine learning for computer vision we present the structure of the proposed *ConvLSTM* based models, their hyperparameters found to work best and more information around their choice. An explanation of the evaluation and performance metrics that we will be comparing our results upon are thoroughly presented. Those metrics include numeric methods, image-based (object detection) and time series related error metrics. We then proceed to compare their effectiveness along various configurations of critical input and output formats, such as the length of the forecasting history, forecasting horizon, and the y-axis size. Explanation and comments on the model's behavior is given, highlighting key observations and the chosen input and output configurations for further analysis.

Chapter 5 briefly reviews traditional time series forecasting methods. However, experiments showed their performance to be significantly lower than the proposed image-based model and consequently, these methods were excluded from further analysis. Subsequently, the chapter dives into the structure, hyperparameters, and selection process of the *Long Short-Term Memory LSTM* based model. This section also analyzes the effectiveness of the LSTM model under various configurations providing the performance metrics for different combinations of forecasting history length and horizon. Additionally, comparison plots are included to contrast the *LSTM's* performance with the previously explored *ConvLSTM* model. Explanations and observations regarding these comparisons are provided. Furthermore, a discussion is presented on the complexity, size, and training times associated with each model. Finally, the models are compared again by testing their behavior on different sections of the dataset, revealing insights into their generalization capabilities of image-based approach.

Chapter 6 summarizes the key findings of this thesis and evaluates their significance in achieving the research objectives. It then discusses the challenges and limitations encountered during the research process. Finally, the chapter and thesis concludes by proposing potential areas for future research, along with an justification for their exploration.

Chapter 2

Background

In this section we are going to present background material, essential for the explanation and full understanding of the methods we have used in the main part of the thesis. First statistical models for time series forecasting are presented and then machine and deep learning approaches, utilized in this work, are described.

2.1 Traditional Forecasting Methods

Statistical time series forecasting is a fundamental approach in data analysis, offering valuable insights into understanding and predicting the behavior of sequential data points over time. Through the utilization of statistical method, such as autoregression, moving averages, etc., these methodologies aim to extract meaningful patterns and dependencies from time series datasets and ultimately aid in the anticipation of future trends, identify anomalies in sequential data.

Two of the most known and widely used approaches in time series forecasting are ARIMA (AutoRegressive Integrated Moving Average) and exponential smoothing. Although these techniques may be applied independently, they provide complementary statistical information. In detail, ARIMA models are based on the calculation of the autocorrelation within a sequential dataset, while exponential smoothing aim to describe the seasonality and the data's trend.

2.1.1 Exponential Smoothing

Exponential smoothing methods provide a flexible approach for forecasting by assigning exponentially decreasing weights to past observations, with an emphasis on recent data. This ensemble of algorithms is ample of forecasting time series, that exhibit trends or seasonality, high accuracy and speed.

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2 y_{t-2} + \dots \quad (2.1)$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter.

As seen in equation 2.1.1, future observations of time series are estimated as the weighted average of the current and the past observations, where the weights decay exponentially the older the observation is. Evidently, the greater the factor is the more emphasis is given to recent observation and vice versa.

2.1.2 ARIMA

Auto Regressive Integrated Moving Average, or in short ARIMA, models are created as a linear combination of simpler Auto Regressive (AR) and Moving Average (MA) models [81]. Auto Regressive models predict future observations as a linear combination of past values of a certain variable, while Moving Average models are employed to model the forecasting error by calculating a linear combination of previous forecasting errors [14, 16, 32].

In an autoregressive model, the output variable y_t depends linearly on its previous values (y_{t-1}, \dots, y_{t-p} and some white noise ϵ_t). A sample y_t is said to be autoregressive of order p , denoted as $AR(p)$, if y_t is described by the following relationship:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t \quad (2.2)$$

, where ϵ_t is white noise and ϕ_1, \dots, ϕ_p are the sample weights

On the other hand, Moving Average models assume that t is a random variable with zero mean and unitary variance, and hence aim to model the forecasting error in an autoregressive manner. This model is referred to as a $MA(q)$, a Moving Average model of order q and explained in the following equation:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_p \epsilon_{t-p} \quad (2.3)$$

, where ϵ_t is white noise and $\theta_1, \dots, \theta_p$ are the sample weights

The combination of the aforementioned fundamental models results in the, so called, $ARMA$ models. $ARMA$ models encapsulate the temporal dependencies in a time series by blending the weighted sum of past observations from both AR and MA components. The AR component captures the linear relationship between current observations and a weighted sum of previous observations, while the MA component models the dependency between the current observation and a stochastic error term based on past errors. An $ARMA$ model, of order p for the AR component and order q for the MA sub-model is symbolized as $ARMA(p, q)$ and described by the following equation:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_p \epsilon_{t-p} + \epsilon_t \quad (2.4)$$

A major shortcoming is that $ARMA$ models require the time series to be stationary in mean and variance. In order to overcome this obstacle the introduction of an Integration component was suggested by Box et al in [81], giving rise to the $ARIMA$ models. In this manner, a time series is differentiated, by the number of times necessary, so that it becomes mean stationary. To calculate the variance stationary equivalent, power transformations or box-cox approaches are utilized [36].

An $ARIMA(p, d, q)$ model is created by combining an $AR(p)$ and a $MA(q)$ model, and by differencing d times the original time series. The entire model can be written as follows

$$y'_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + \dots + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_p \epsilon_{t-p} + \epsilon_t \quad (2.5)$$

Where: y'_t is the value at time t , c is a constant, ϕ_i (for $i = 1, 2, \dots, p$) are the autoregressive coefficients, θ_i (for $i = 1, 2, \dots, p$) are the moving average coefficients and ϵ_t is the error term at time t .

2.2 Deep Learning

Neural networks consist of multiple layers of artificial neurons, which receive a vector input, calculate its linear combination, and subsequently apply a non-linear activation function, e.g. *Sigmoid*, *Hyperbolic Tangent*, *Rectified Linear Unit (ReLU)*, etc, to produce their outputs. The outputs of neurons in one layer are forwarded as inputs to the neurons in the immediately subsequent layer of the network and correspond practically to simple features of the network's input. In this way, deep neural networks, or *Feed-Forward Neural Networks (FNN)*, gradually learn to combine simpler features through the *backpropagation* algorithm to compose new, higher-level features until the information reaches the network's output, which represents a complex non-linear transformation of the network's input (Figure 2.1). This deep architecture of modern neural networks imbues them with strong capabilities for abstraction and generalization.

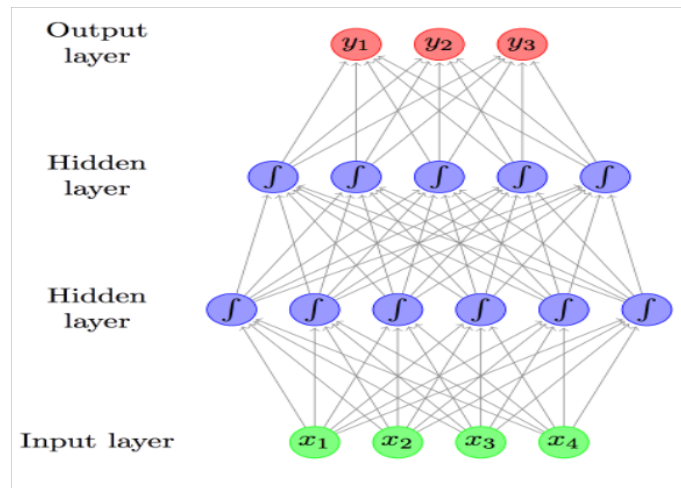


Figure 2.1: Neural Networks Core Structure

2.2.1 Recurrent Neural Networks and Long Short-Term Memory Networks

FNNs, although capable of identifying non-linear correlations and dependencies in tabular data, they fail to model temporal dependencies that are present in sequential datasets, like time series. While they utilize purely non-linear activation functions to learn complex patterns, their architecture lacks the inherent memory mechanism necessary for capturing sequential information effectively. In this manner, a special form of artificial neural networks has emerged, the *Recurrent Neural Networks (RNN)*. *RNNs* are designed to effectively model sequential data by incorporating feedback loops within their architecture, allowing them to maintain a form of memory. Key components include hidden states that retain information about previous inputs, enabling *RNNs* to capture temporal dependencies in data [72].

Numerous variants of *RNNs* have been proposed in literature, depending on the number of inputs and outputs the process and generate respectively. The four most prominent

ones are: one-to-one, which is similar to traditional feed-forward networks in that there is only one input and one output; one-to-many, in which a single input can result in multiple outputs; many-to-one, in which numerous inputs from various time steps result in a single output; and many-to-many, in which numerous inputs result in numerous outputs (Figure 2.2).

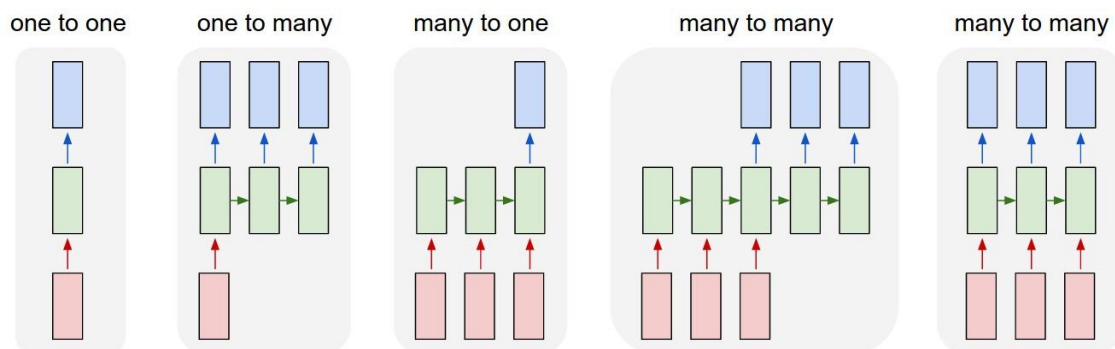


Figure 2.2: The different types of Recurrent Neural Networks.

Conventional *RNNs* are ample of processing effectively sequential data of with rather low temporal resolution, i.e. relatively short time series, however when analyzing longer sequential data the problem of vanishing gradient arises [35]. To overcome the problem of gradients' values become very small (vanishing gradient problem) during the learning process of such models', various sophisticated architectures have been proposed. Historically, architectures like *Bidirectional Recurrent Neural Networks (BRNN)*, *Gated Recurrent Unit (GRU)* and *Long Short Term Memory Networks (LSTM)* have been in the spotlight [53, 20].

LSTMs are the most commonly used models for sequential data analysis [29], hence are employed in this thesis and further explained in the following subsection.

LSTM

Long Short Term Memory Networks (LSTMs), are a specialized type of *RNN* known for its ability to capture long-term dependencies and mitigate the vanishing gradient problem often encountered by traditional *RNNs*.

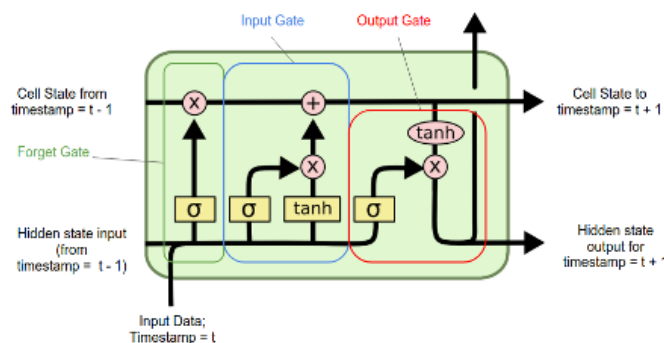


Figure 2.3: The different types of Recurrent Neural Networks.

The architecture of an *LSTM* consists of three crucial layers: the "forget gate layer", responsible for determining whether to retain or discard information from the previous

timestamp; the "input gate layer", which integrates new information into the cell; and the "output gate layer", facilitating the passage of updated information to the next timestamp. Additionally, *LSTMs* maintain a hidden state (h_{t-1} representing the previous timestamp's hidden state and h_t representing the current timestamp's hidden state) and a cell state, c_{t-1} and c_t for the previous and current timestamps, respectively, to preserve memory across sequential data points.

2.2.2 Convolutional Neural Networks

As aforementioned, *FNNs* excel in analyzing tabular data that do not exhibit neither temporal nor spatial correlations. In a similar manner with *RNNs*, *Convolutional Neural Networks (CNN)* were designed to effectively utilize images as their input, i.e. 2-dimensional arrays whose elements are highly spatially correlated. This specific architecture, following the example set by the *FNNs*, consists of neurons with specific weight and bias values, while the output of each layer is further passed through an activation function, thus introducing the required non-linearities in the models output.

The key difference between *CNNs* and conventional *FNNs* is explicit assumption made, that the network's inputs are purely 2D arrays. In this manner, single-valued weights in each neuron are substituted by 2D square kernels, thus drastically reducing the network's number of parameters needed to sufficiently model the non-linear relations amongst an image's pixels. Moreover, in *CNN* instead of generating the linear combination of each neurons inputs, the convolution of each neuron's input with the respective kernels is calculated [48]. Subsequently, the non-linear activation function of each layer, e.g. *hyperbolic tangent*, *ReLU*, is applied element-wise to each neuron's output.

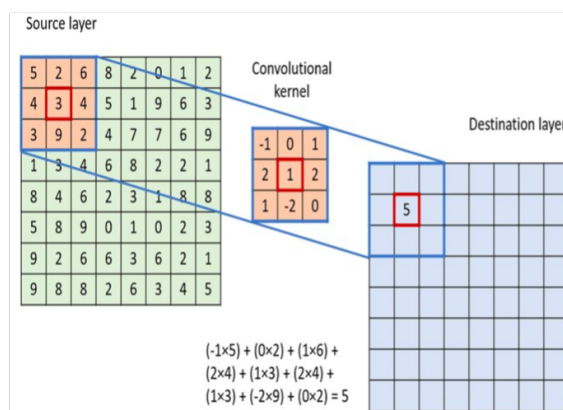


Figure 2.4: The application of a convolutional kernel within a CNN architecture.

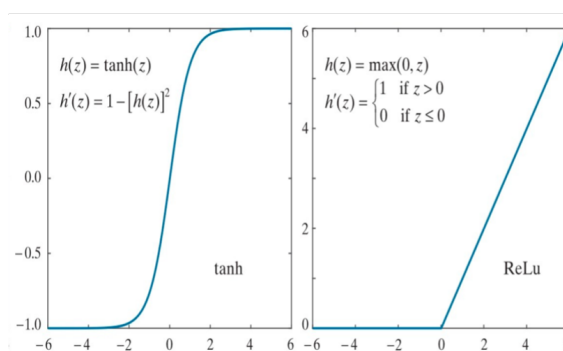


Figure 2.5: *Sigmoid* and *Relu* activation functions.

Through their forward propagation, 2D data arrays are analyzed in various scales by the alternating in size convolutional kernels of the *CNN*. In this manner, spatial features of various scales and detail are extracted constituting *CNNs* a powerful and efficient algorithm for image processing and the generation of dense, non-linear representations of the original data.

2.2.3 Convolutional Autoencoders

Autoencoders are neural networks designed for unsupervised learning, aiming to efficiently compress and reconstruct input data. Consisting of an encoder, $E(x)$, and decoder $D(x)$, two mirrored neural networks, they learn to encode the input into a lower-dimensional latent space and decode it back to the original form [83]. Widely used for tasks like data denoising, dimensionality reduction, and feature learning, autoencoders play a crucial role in extracting meaningful representations from complex datasets without the need for labeled training examples [11].

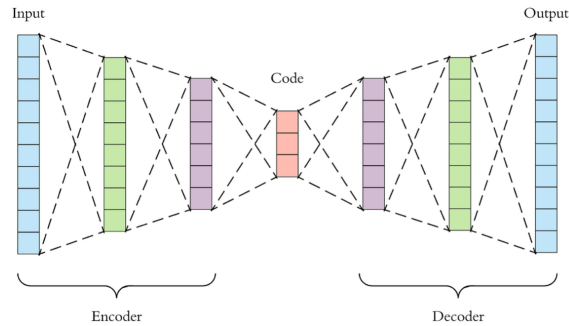


Figure 2.6: High-level architecture of an autoencoder.

Similarly with the *FNNs*, to enhance the capabilities of an *Autoencoders*, it has been suggested to substitute the fully connected layers with convolutional layers. In this manner, *Convolutional Autoencoders* are ample to analyze images or other 2D data arrays and generate meaningful, dense representations of the originals images. Within the architecture, the max pooling layer, *Relu* activation functions, and convolutional layer often constitute the encoder [91, 18]. In order to link to the latent space, the output of this layer is flattened and fed into a completely connected layer. The encoder network is replicated in the decoder.

The convolutional autoencoder will be utilized in this thesis to create a visual forecast image from an input time series image by learning an incomplete mapping $D(E(x))$, in which the decoder network $D(\cdot)$ reconstructs the forecast image from the embedding vector and the encoder network $E(\cdot)$ learns meaningful patterns and projects the input image x .

2.2.4 Convolutional Long Short-Term Memory Networks

Convolutional Long Short-Term Memory networks (ConvLSTMs) are a specialized form of *RNNs* designed to handle sequential data with spatial structure, such as videos or time-series data. Unlike traditional *LSTMs* which completely rely on sequential analysis of data, *ConvLSTMs* incorporate convolutional operations to analyze consecutive frames or segments simultaneously, thereby capturing both spatial and temporal dependencies within the data. This enables *ConvLSTMs* to effectively model complex patterns and dynamics present in sequential data by taking advantage of the inherent spatial relationships across series of frames.

For spatiotemporal prediction, *ConvLSTM* processes a grid cell's future state based on inputs and the historical states of its nearby neighbors. A straightforward method to do

this is to use a convolution operator in the input-to-state and state-to-state transitions [74]. Put otherwise, it functions as a recurrent layer in the same manner as the *LSTM*, except instead of internal matrix multiplications, convolution operations are used.

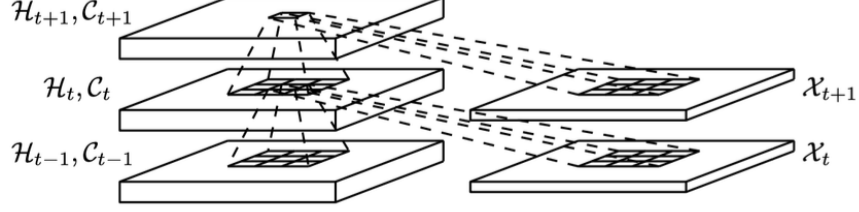


Figure 2.7: Inner structure of *ConvLSTM* [74].

Relu activation layer and Max-Pooling layers might be placed on top of and coupled with the *ConvLSTM* layer. As a result, Autoencoder-based models that resembled those shown in the preceding subsections could be constructed. Specifically, a *ConvLSTM* layer is used in place of a convolutional layer. This method allows the model to record sequential and geographical information.

Finally, the key equations of *ConvLSTM* are shown below, where $*$ denotes the convolution operator and \circ the Hadamard product [74]:

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \quad (2.6)$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \quad (2.7)$$

$$c_t = f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \quad (2.8)$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \quad (2.9)$$

$$H_t = o_t \circ \tanh(C_t) \quad (2.10)$$

Chapter 3

Visualization of Data Access Patterns

This chapter introduces the dataset and explores the preprocessing steps followed.

3.1 Dataset Description

The data is designated for an "ML Prefetching Competition" during the ML For Computer Architecture and Systems Workshop (MLArchSys) [2] that takes place in the International Symposium on Computer Architecture (ISCA). The dataset provides traces for numerous memory-intensive benchmarks with the aim of evaluating machine learning models for data prefetching. The trace files consist of a sequence of memory loads at a 64-byte level which are collected at the L3 level, so the stream is filtered by the L2 cache.

3.1.1 Dataset Source

The traces come from **SPEC 2006** [3, 34], **SPEC 2017** [4, 15] and **GAP** [13] benchmark suites. We will proceed by explaining what are these benchmarks used for, how are they created and what is their importance.

Because a computer system's performance can't be characterized by one number or one benchmark the Standard Performance Evaluation Corporation (SPEC) was created, as a non-profit corporation formed to ascertain, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the most recent generation of computing systems. These CPU benchmark suites are commonly used in computer architecture research and become mainstream for simulation-based design and optimization for next-generation processors, memory subsystems, and compilers. SPEC designed this suite to produce a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real user applications. The SPEC CPU 2006 and 2017 benchmarks have several ways to evaluate computer performance. One way is to measure the time needed for the PC to complete one task; this is a speed measurement or by counting how many tasks can be completed by the computer in a specific time; this is a throughput measurement. The SPEC 2017 suite was the replacement for the SPEC 2006 suite. To help keep the benchmarking data honest, fair, and relevant, SPEC draws benchmarks from real life applications, rather than using artificial loop kernels or synthetic benchmarks. Some examples of the benchmarks and their application are depicted on the Table 3.1 where we can observe that the application area of the benchmarks come from real world use cases.

SPEC 2006 consists of 30 benchmarks and SPEC 2017 from 43 benchmarks. In our dataset (provided from the MLArchSys Workshop) we are given more than one trace file per

Benchmark Name	Application Area
500.perlbenc	Perl interpreter
505.mcf	Route planning
520.omnetpp	Discrete Event simulation - computer network
549.fotonik3d	Computational Electromagnetics
554.roms	Regional ocean modeling

Table 3.1: Example of benchmarks included in SPEC2017 suite and a brief explanation of application area

benchmark in most cases, in total we are given 32 trace files from 11 different benchmarks from SPEC 2006 as shown bellow:

- 410.bwaves
- 437.leslie3d
- 429.mcf
- 473.astar
- 462.libquantum
- 450.soplex
- 470.lbm
- 482.sphinx
- 433.milc
- 459.GemsFDTD
- 471.omnetpp

and 47 trace files from 9 benchmarks from SPEC 2017:

- 602.gcc
- 620.omnetpp
- 649.fotonik3d
- 605.mcf
- 621.wrf
- 654.roms
- 619.lbm
- 623.xalancbmk
- cactuBSSN

Graph abstractions have been in use for decades, but in the last decade it’s value has grown, particularly as social networks and their analysis has become more significant; graph algorithms have also been used in applications in science and recognition. As a result, there is a great interest in well-supported graph processing. Given that, with the intention of promoting a standardization of the graph processing evaluation a set of benchmarks for graph processing was created, the *GAP Benchmark Suite*. The latter creates less conflicts between graph processing evaluations and will make it easier to compare between different research efforts and to quantify improvements. The benchmark specifies input graphs, graph kernels, and measurement techniques. The suite also features a baseline reference implementation that has been optimized to represent state-of-the-art performance. This benchmark collection can be applied in a number of contexts. By implementing all of the benchmark’s kernels and providing competitive performance on all of the benchmark’s graphs, developers of graph frameworks can show the generality of their programming approach. The reference implementations and the input graphs can be used by algorithm designers to showcase their work. The suite can also be used by performance analysts and hardware platform designers as a workload model for graph processing.

GAP benchmark suite consists of 30 tests based on 6 graph kernels on five graphs each. The kernels were selected based on how commonly they are used. These kernels are indicative of several applications in science, engineering, and social network analysis. Due to the inclusion of both traversal-centric and compute-centric kernels, this collection of kernels exhibit computational diversity. Throughout the suite, several kernels take into account or disregard certain graph features, such as edge weights and edge directions. A brief explanation of the kernels can be seen on Table 3.2.

The five input graphs selected are different in topology and origin (synthetic or real-world).

Kernel	Explanation
Breadth-First Search (BFS)	Traverses all vertices at the current depth (distance from the source vertex) before moving onto the next depth
Single-Source Shortest Paths (SSSP)	Calculates the shortest path from a given source vertex to every other reachable vertex. The distance between two vertices is the minimum sum of edge weights along a path connecting the two vertices
PageRank (PR)	Computes the PageRank score for all vertices in the graph
Connected Components (CC)	Labels all vertices by their connected component and each connected component is assigned its own unique label.
Betweenness Centrality (BC)	Approximates the betweenness centrality score for all vertices in the graph by only computing the shortest paths from a subset of the vertices. The betweenness centrality of a vertex v is defined to be the fraction of shortest paths that pass through it
Triangle Counting (TC)	Computes the total number of triangles in a graph

Table 3.2: Graph Kernels of GAP Benchmark Suite

Real-world data, models the relationship between individuals, websites and roads. The data collected consists of graphs that are among the most accessible in terms of licensing requirements and bandwidth availability making it simple for users to receive them. The graphs chosen can be seen at Table 3.3. In our dataset we are given 4 trace files for the BFS, SSSP, PR, CC and BC kernels, thus in total we have 20 trace files.

3.1.2 Raw Data Format

Each trace file includes a sequence of memory loads at a 64-byte level of granularity where the stream is filtered by the L2 cache giving as a memory trace collected at the L3 Level. The trace files are given in a comma separated regular text file (".txt"), where each row

Input Graph	$ V $	$ E $	Explanation	Origin
Twitter	61.6M	1468.4M	Social Network topology	Real-world
Web	50.6M	1949.4M	Web-crawl of the .sk domain (sk-2005)	Real-world
Road	23.9M	58.3 M	Distances of all of the roads in the USA	Real-world
Kron	134.2M	2111.6M	Kronecker graph generator	Synthetic
Urand	134.2M	2147.4M	Generated by the Erdos Reyni model	Synthetic

Table 3.3: Input graphs of GAP Benchmark Suite. $|V|$ is number of vertices and $|E|$ number of Edges of graph

First five rows of bc-0.txt trace file				
14,	152,	28e837c89f00,	14ce07e1e230,	0
17,	169,	28e837c8af40,	14ce07e1e240,	0
18,	509,	28e837c8c840,	14ce07e1e247,	0
19,	633,	28e837c8c740,	14ce07e1e24b,	0
22,	634,	28e837c8c7c0,	14ce07e1e254,	0

Table 3.4: First five rows of bc-0.txt trace file. Where the comma separated values of each row correspond to a load memory access instruction and the values from left to right are the Unique Instruction Id of the load, CPU cycle timestamp, Physical address of the memory request, Instruction Pointed of the load (PC) and Last Level Cache (LLC) hit/miss (1/0)

consists to a load memory access and corresponds the information as follows:

- Unique Instruction Id of the load Instruction
- CPU cycle timestamp for the load
- Physical address of the memory request
- Instruction Pointed of the load (PC)
- Last Level Cache (LLC) hit (1) or miss (0)

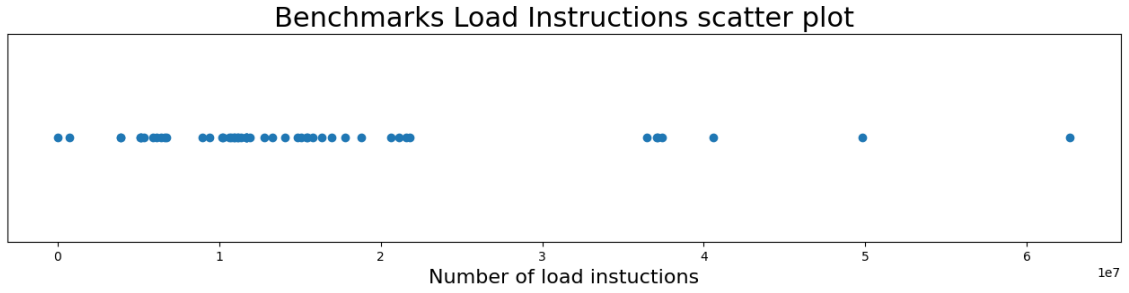
An example from the first rows of the trace file *bc-0.txt* can be seen on Table 3.4.

In total we have 99 trace files, each one of these trace files have a large number of lines, load instructions. As we can see from Figure 3.1 (plots (a) and (b)), most benchmarks have a number of instructions between $1 * 10^7$ or $2 * 10^7$. We can also see the rough distribution between hit and misses from the Figure 3.2 where we plot for each suite of benchmarks the count of total load instructions, count of how many were hits and how many were misses for each benchmark (we take the mean value for trace files of the same benchmark). At this point we should not forget that we are working with the memory trace file of the LLC, given that we have no interest at the load instructions that *hit*, meaning there where found in the LLC, but with the instructions that were *miss* and not found in the LLC. From Figure 3.1 (plots (c) and (d)) again we can see the distribution of the number of entries of *miss* instructions for all the trace files. We can see that we have around $2.5 * 10^6$ instructions.

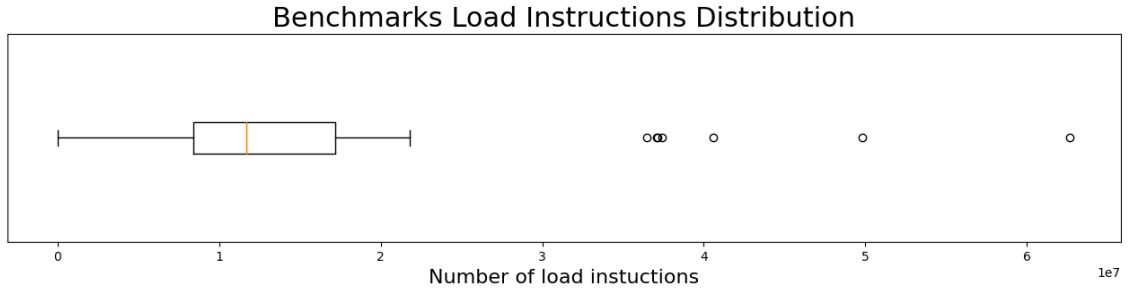
3.2 Image Based Pipeline

3.2.1 Input Transformation/Data Preprocessing

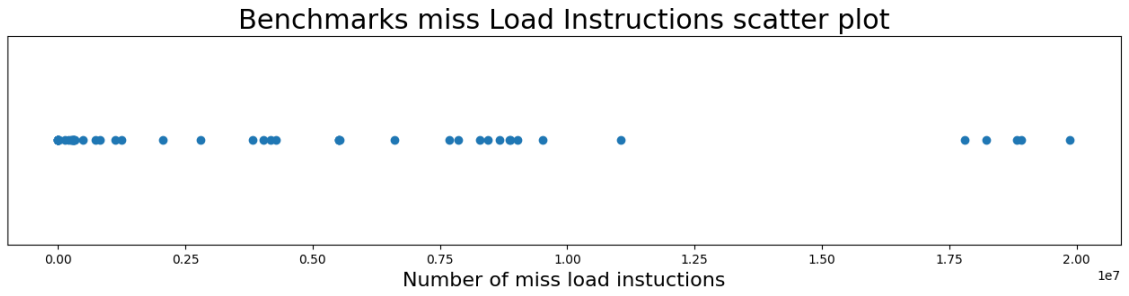
One of the most famous memory management techniques of *Operating Systems (OS)* is paging. From *OS* of the 60s up to modern *OS* of today, almost all of them use a page size of 4 KB [86] thus we are working on a page granularity of 4 KB. The given trace files provide us the physical address as a hexadecimal (base 16 numbering system) number, the third value of each row from Table 3.4. Taking all the addresses of the *miss* instructions we have a sequence $\{h_0, h_1, \dots, h_t\}$ where h_t a hexadecimal number. In our pipeline, we take each hexadecimal address h_t and convert it to the corresponding decimal (base 10 numbering system) value giving us a sequence $\{d_0, d_1, \dots, d_t\}$ where d_t a base 10 integer number. Then we take each address d_t and find the closest integer divisible by 4096 (the page size we chose) giving us the base address dp_t . This way we correspond a *range* of hexadecimal addresses to decimal address . Then we use these dp_t addresses as a key to



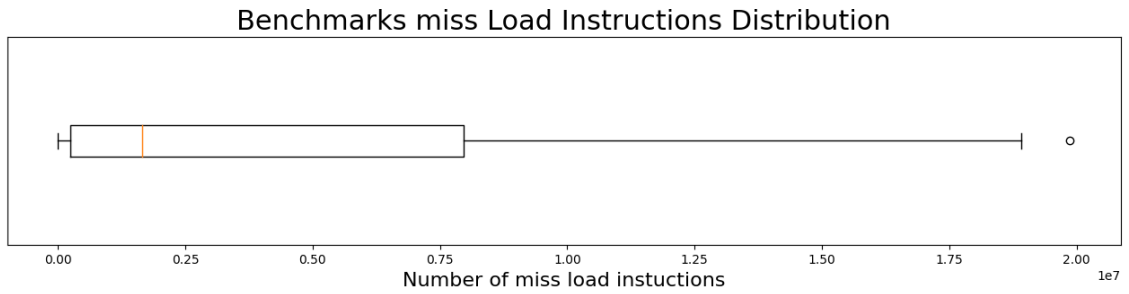
(a)



(b)



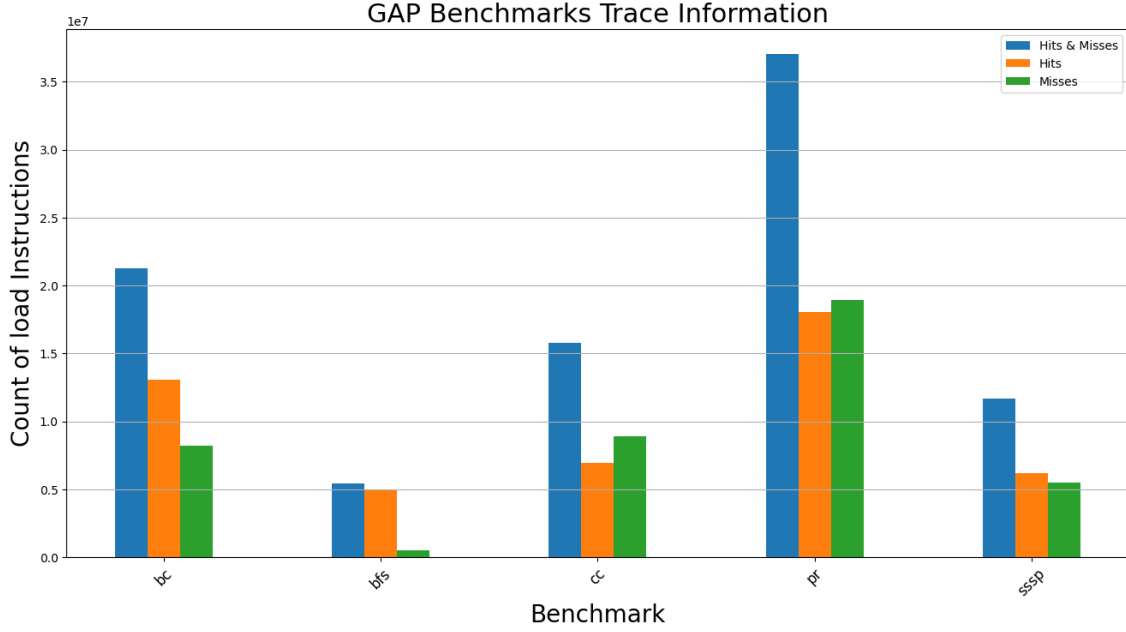
(c)



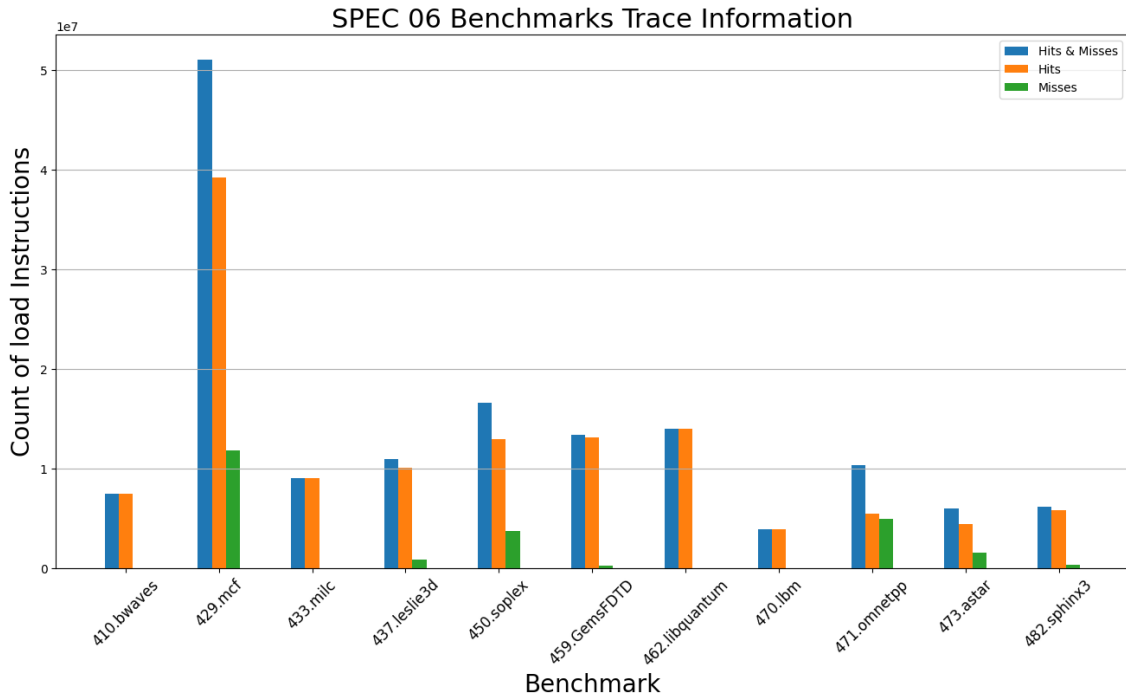
(d)

Figure 3.1: Plot of the total number of load instructions of all benchmarks (a) scatter plot of the number of load instructions for each benchmark (b) box plot of the number of load instructions for each benchmark (c) scatter plot of the number of *miss* load instructions for each benchmark (d) box plot of the number of *miss* load instructions for each benchmark

map each one of them to an integer y_t starting from zero and adding one with each new key-address we encounter. If we come across a $dp_{t'}$ address where $dp_{t'} = dp_t$ then we have the mapping to find the corresponding $y_{t'} = y_t$ integer value. An example pipeline can be seen in Figure 3.4. Repeating the above for all of the *miss* $\{h_0, h_1, \dots, h_t\}$ values of a trace file we end up with a 1D time-series of integer values $\{y_0, y_1, \dots, y_t\}$ where $y_t \in \mathbb{Z}^+$. From

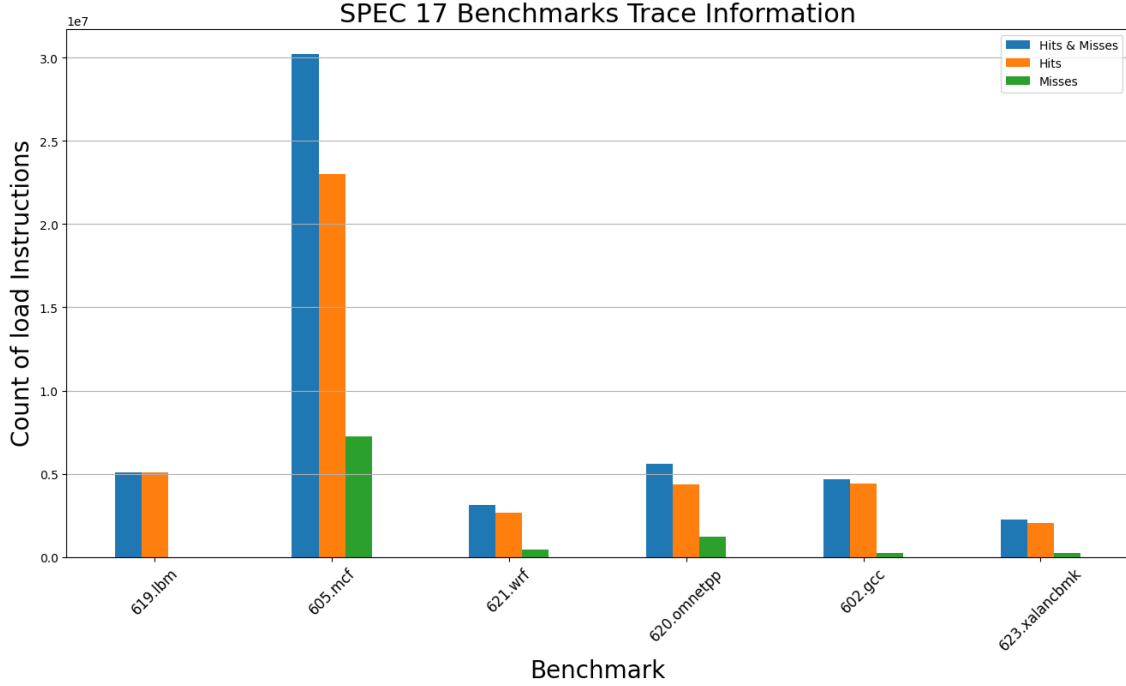


(a)



(b)

the row in the trace file of which we took the h_t value, the second value of that row (see Table 3.4) gives us the *CPU cycle timestamp* or else the *cycle count*, thus for h_t we have the corresponding cc_t which is an increasing integer number as we go through the trace file. Consecutively cc_t describes the *cycle count* value for y_t too. We must note at this point that for the time-series created above, the sequence of *cycle count* cc_t and page range value y_t pairs (cc_t, y_t) , is not equally spaced. Meaning that the difference $\Delta cc_{t,t-1} = cc_t - cc_{t-1}$ is not constant for all cc_t, cc_{t-1} of the specific trace file.



(c)

Figure 3.2: Plot depicting the hits and misses of load instruction for each benchmark in each suite. (a) for GAP benchmark (b) for SPEC 2006 and (c) for SPEC 2017.

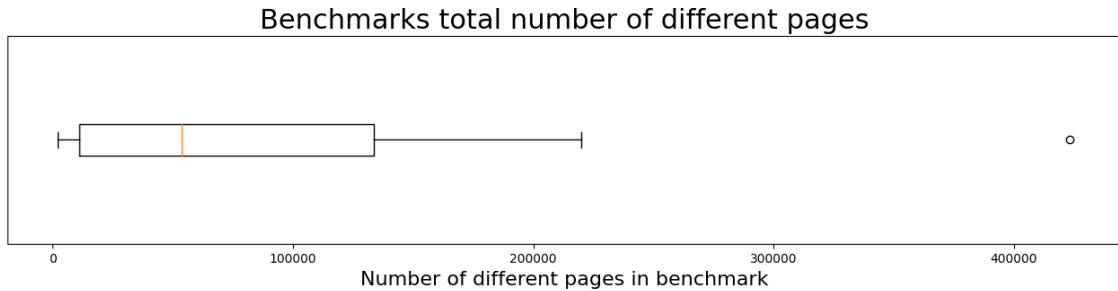


Figure 3.3: Plot depicting distribution of the total number of different pages accessed by all benchmarks

3.2.2 Data Visualization

To make an accurate prediction from a 2D image, you first need a good 2D visualization of the subset of time-series $\{y_0, y_1, \dots, y_t\}$. Otherwise no algorithm will be able to understand the fundamentals of the data and make accurate predictions if the image does not represent the series main pattern.

Based on previous work, Recurrence Plots [51] and GAF [12] are common transformation approaches that have been used in the field to visualize time-series as an image ($\{y_0, y_1, \dots, y_t\} \rightarrow I_t$). These methods are capable of mapping a time-series to an image with rich information and have been proven great for various tasks, unfortunately though there still is no way to map back the image to the time-series ($I_t \rightarrow \{y_0, y_1, \dots, y_t\}$). Given that, because we are using an image-to-image approach and eventually are seeking to revert to a time-series result the representation of the time series as a recurrent plot or GAF is not suitable.

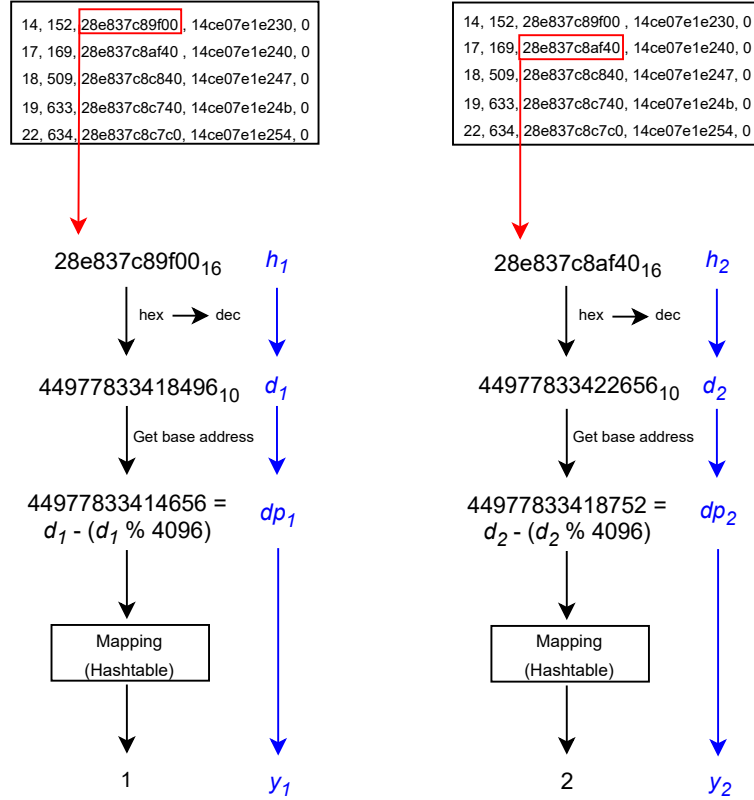


Figure 3.4: Pipeline example for the first 2 inputs for a new trace file execution.

Based on the work of Veloso et al.[19] we are going to work with a simple approach of visualizing the time-series. Before explaining the visualization we ended up using we will go throughout the process and some other possible solutions we came up with and eventually the one we ended up choosing as method. As mentioned earlier we are given a time-series $\{y_0, y_1, \dots, y_t\}$ where $y_t \in \mathbb{Z}^+$, this time-series can have a length of around $2.5 * 10^6$ and also the range of y_t values are around Y (for a distribution of the total number of different pages see Figure 3.3). At first for plotting the above time-series we use python library `matplotlib` which automatically scales the input series to an image of specified size, some plots (of the whole dataset) can be seen in Figure 3.5. Also we must note, that binary images (black or white valued pixels only) were created, more discussion about this decision will be done later. Seen from the examples plots of Figure 3.5 some plots show some consecutive *strides*, meaning pages that are repeated consecutively throughout the execution of the benchmark. Furthermore we can observe that the plots seem to be "dense" in information, others throughout the whole x and/or y axis and others on specific parts of the image. This happens because on the x-axis we are compressing a couple million of points to a few hundred of pixels and similarly on the y-axis we have tens of thousands of pages corresponding to a couple hundred pixels too. The latter means that we have a huge amount of conflicts of data pairs (cc_t, y_t) and $(cc_{t'}, y_{t'})$ ending up at the same pixel of the image. The latter images of course are not acceptable for our needs because we have a great loss of information due to the thousands of points overlapping each other.

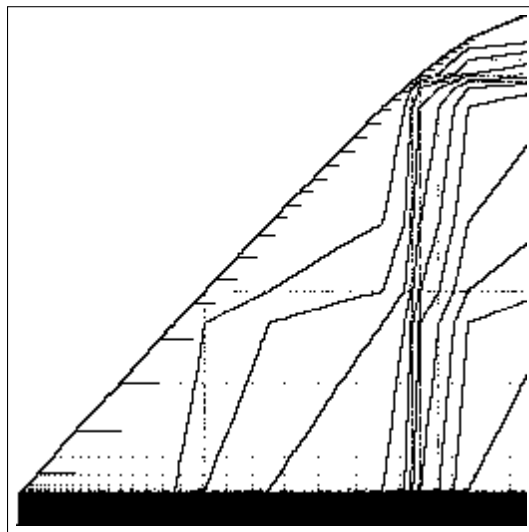
While exploring the image visualization techniques and how to depict the information more precisely we found some rather interesting aspects. In plenty of the benchmarks, as expected, due to the vast "compression" that has been done to confine all their data to image of a couple hundred pixels in each axis, a huge amount of valuable information is lost. We proceed by *zooming* on parts of these plots. By *zooming* we mean depicting on an

image less information ((cc_t, y_t) points) proportionate to the image dimensions (in pixels). We can depict a smaller part of the dataset on either the x-axis (by keeping the points in a specific Δcc range) either the y-axis (by keeping the points in a specific page range Δy) or both. Eventually producing a higher resolution image for the specific area. Two examples can be seen in Figure 3.6 where we can see that by gradually zooming in we "discover" new *strides* of memory accesses, of course that is not true for all cases as can be seen in Figure 3.7. Naturally by emphasizing on a specific area by *zooming* to depict the whole information of the benchmark we need more than one images.

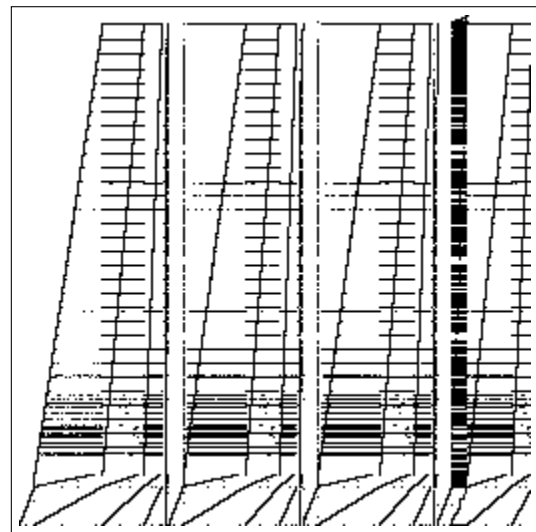
By exploring the behavior of different benchmarks we found some interesting cases. Where there is a large amount of information/points that is condensed in a specific page range (Δy) it leads, when plotting the whole dataset, to a black rectangle throughout the image. We describe this scenario as *Black Bar* benchmarks, such benchmarks are the ones listed below:

- bc
- cc
- 605.mcf
- sssp
- pr

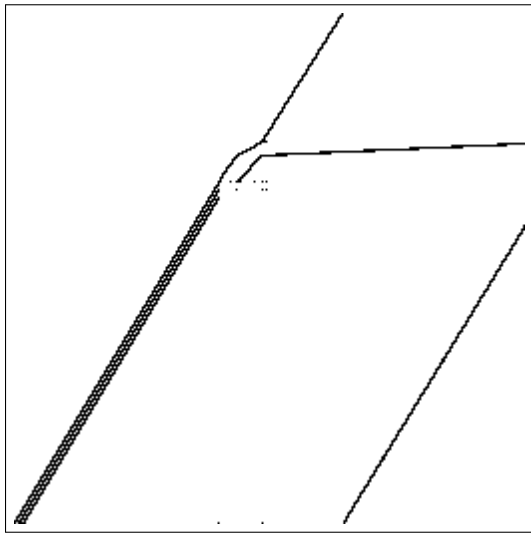
We can see two examples from figure 3.5 plots (a) bc-0 and (f) cc-5 where the "*Black Bar*" is clearly visible. Zooming in these sections, where the page values range is considerably smaller, a lot of *strides* start to be visible.



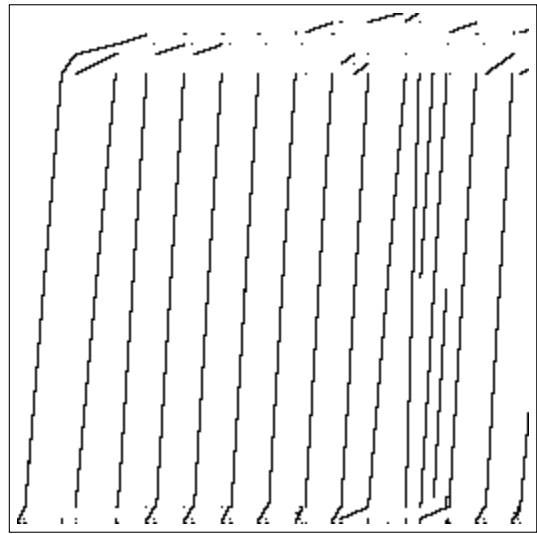
(a)



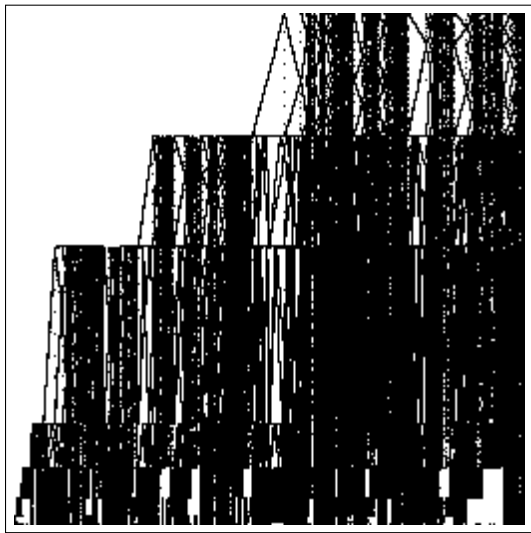
(b)



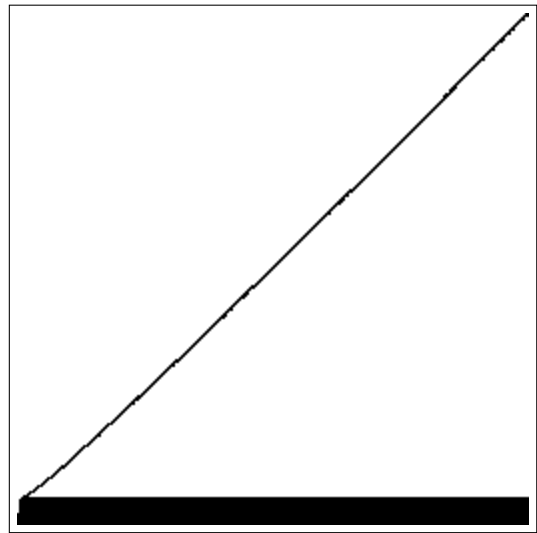
(c)



(d)

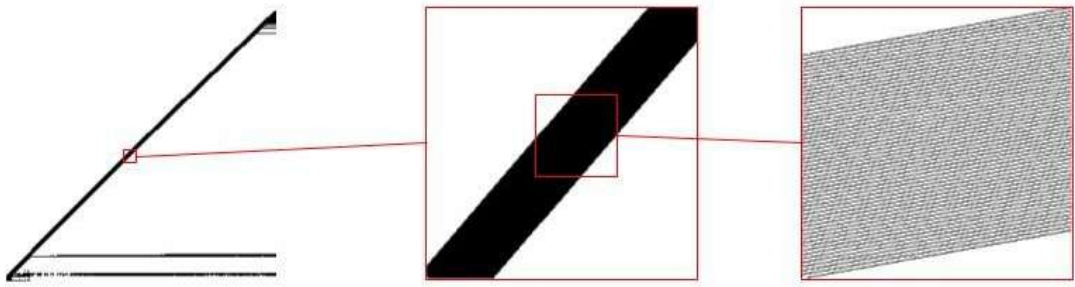


(e)

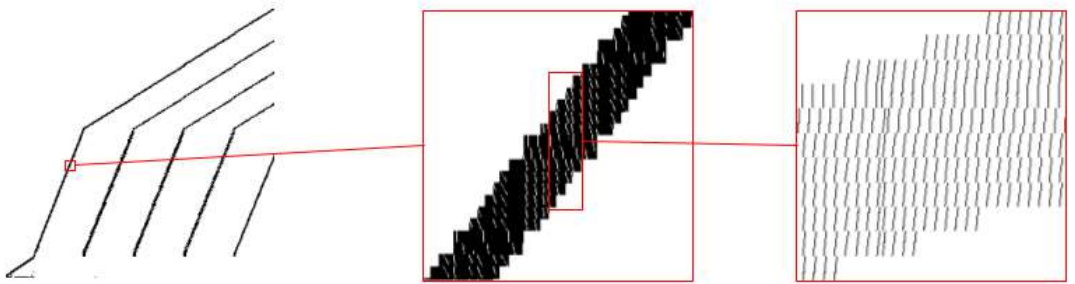


(f)

Figure 3.5: Plots using `matplotlib` for (a) `bc-0` (b) `bfs-10` (c) `410.bwaves-s0` (d) `433.milc-s1` (e) `605.mcf-s8` (f) `cc-5` trace files .

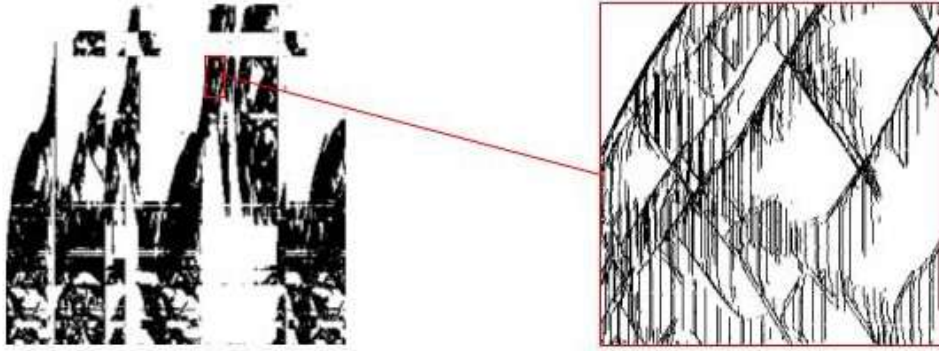


(g)

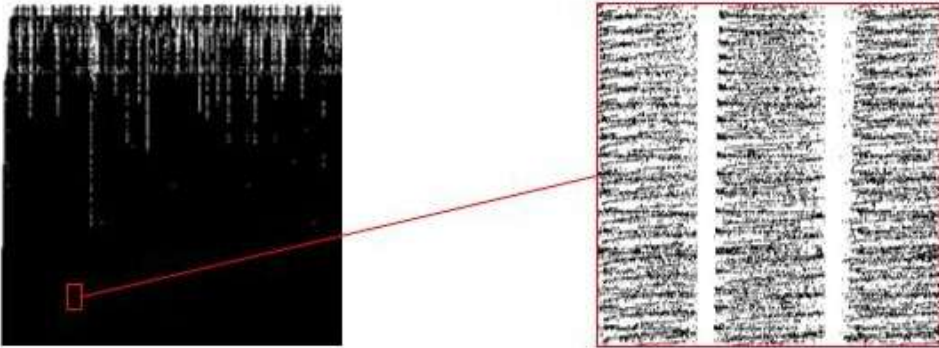


(h)

Figure 3.6: Examples of *zooming* on benchmarks (a) 605.mcf-s1 (b) cactuBSSN-s0 .



(a)



(b)

Figure 3.7: Examples of *zooming* on benchmarks (a) 473.astar-s1 (b) 473.astar-s2 .

"Black Bar" Analysis

We proceed to further investigate the specific benchmarks mention above and more precisely the area of the "Black Bar". A great amount of effort was given on finding a proper depiction of the images. We will briefly go through the possible solutions we came up with and tried, but further analysis of them will not be given as it is out of the scope of this thesis.

As we saw from the Figure 3.6, by zooming in, *stride* like patterns start to become more visible. For all the benchmarks, we kept a Δy page range that was "wide" enough for all pages of the "bar" to fit in it. By examining the plots of the benchmarks we find these ranges, which can be seen in Table 3.5, all benchmarks have a range from 1200 to 2200 pages which is significantly less than the total pages used by the benchmark (which can be seen on the last column of the Table 3.5). A similar methodology in order to decrease the address space significantly can be seen in [33]. Our images cannot be particularly large due to complexity reasons; otherwise, the model parameters and computations will escalate exponentially, making training and inference of our models much more time demanding. Given that, we will be working with photos ranging in sizes of 64 to 512 pixels on the y-axis. The latter decision means that each pixel on the y-axis corresponds to a smaller range of page values δy , where $\delta y = \Delta y / (\# \text{ y-axis pixels})$. For example for the *sssp* benchmark we have $\Delta y = 1300 - 0 = 1300$ which, for an image with a size of 128 pixels on the y-axis, gives

us $\delta y = \frac{1300 - 0}{128} \approx 10$ pages/pixel; meaning that each pixel on the y-axis corresponds to approximately 10 pages. The pixel position value where a y_t page corresponds to in the final image will be denoted with \bar{y}_t . Figure 3.9 depicts the process of calculating the \bar{y}_t value with an example.

<i>Black Bar</i> benchmarks page range of interest				
Benchmark	Starting Page: y_t	Ending Page: y'_t	# Misses	Coverage (%)
bc	0	2200	11.75M	89
sssp	0	1300	4.5M	70
cc	0	1200	6M	79
pr	3000	4200	14M	79
605.mcf	0	2200	8M	30

Table 3.5: Page range of *Black Bar* part of benchmarks, total number of misses in given page-range and their coverage compared to all misses of benchmark

From this point and further when we are referring to benchmarks we mean the five (bc, cc, 605.mcf, sssp, pr) and only the pages within the page range given in Table 3.5 for each one of them.

In our effort for a good prediction we tried multiple different Δcc ranges or just a different number of points per image. Taking different Δcc ranges has an effect on how many points are kept for the image created as well. In our process to quantify how good the resulting image is, various metrics were used and experimented with but the first main criterion we quantified upon was the visual perception of the strides.

Taking a Δcc means that each pixel corresponds to a smaller δcc (as with Δy and δy) where $\delta cc = \Delta cc / (\# \text{ x-axis pixels})$. By using a big enough Δcc , but still significantly smaller than the whole benchmark size, we still have multiple points ending up in the same δcc and thus the same pixel in the x-axis. Given that we either have points ending up at the same pixel on the x and y axis or multiple values with the same x-axis value.

Having multiple points end up at the same pixel gave us the idea of using gray-scale images and have each pixel value ("grayness" value) depending on the number of points that correspond to that position. The latter was not used because of the added complexity and the problems related to reverting the information from the image back to time-series data. Eventually thought as with the gray-scale images and with other image transformations, one of our tasks is to be able to return back to the original time series or as close as possible. For that reason the idea of having a Δcc was dropped and the images used are one-by-one point depiction of data-points to the x-axis. Meaning that each time-series value corresponds to one point/pixel on the x-axis. Because all cc_t values are unique and we just concluded that we will be working with each one of the points separately one-by-one without considering the difference between two cc_t and cc_{t+1} values we can from now on drop the cc value from our timeseries. Thus giving us a sequence $\{\bar{y}_1, \bar{y}_2, \dots, \bar{y}_t, \bar{y}_{t+1}, \dots\}$. Each \bar{y}_t value will correspond to one column of our image, where on each column only one pixel (row of it) will have a different value from the rest.

Of course our images as stated before are of a specified size from 64 to 512 pixels, but our benchmarks have tens of thousands of load/store miss values. Consecutively, as in the case of *zooming*, to visualize the whole benchmark we need more than one images, a number closer to the tens of thousands of images.

Figures 3.8 and 3.9 depict a visual representation of the above in an example with images of size 10×10 .

Eventually our datasets consists of:

- Only the benchmarks bc, sssp, cc, pr and 605.mcf.
- Only points where the page value is within the ranges given in Table 3.5.
- One-by-one representation of each point to a specific pixel on the x-axis, giving us thousands of images per benchmark.
- Sizes on each axis ranging from 64 to 512 pixels.
- Binary images.

Value: $y_t = 254$

$$\delta y = \frac{\Delta y}{\# \text{ y-axis pixels}} = \frac{1000}{10} = 100$$


$$\bar{y}_t = \lfloor \frac{y_t}{\delta y} \rfloor = \lfloor \frac{254}{100} \rfloor = \lfloor 2.54 \rfloor = 2$$


Figure 3.8: Example of transforming a data point to a corresponding image of 1×10 size

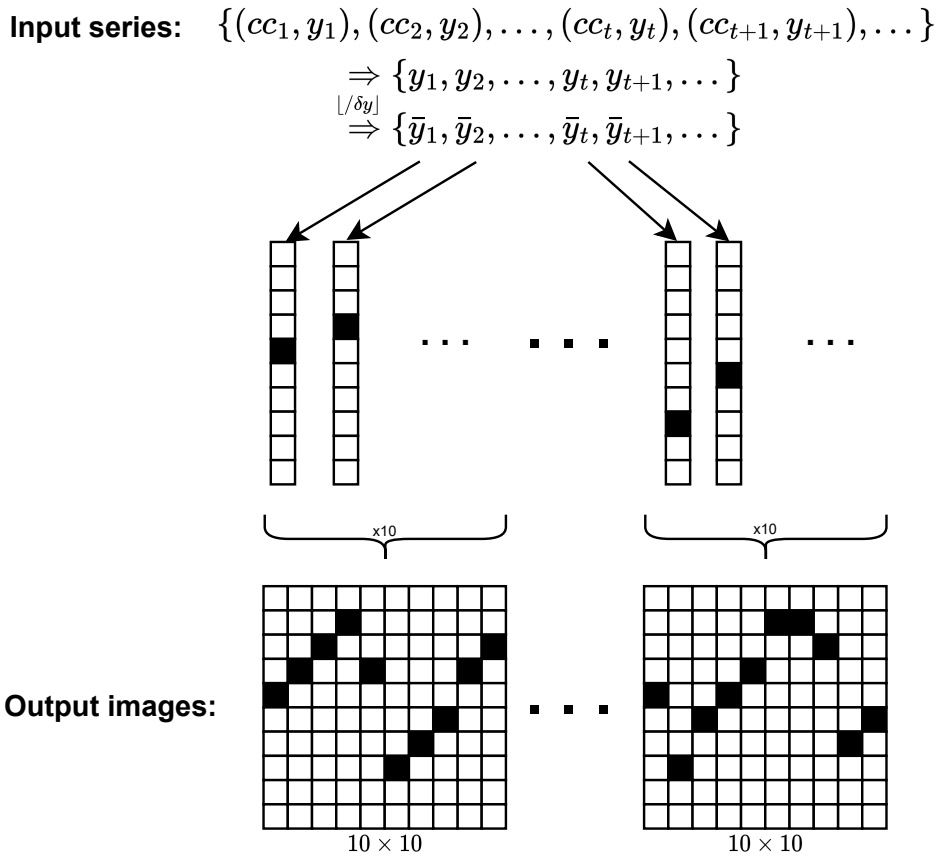


Figure 3.9: Simplified diagram depicting the visualization process of the input time-series to 10x10 images.

3.2.3 Image Decomposition

As stated earlier, we are working with image-to-image models. So given an image, which the creation we explained in the previous sub-chapter, our models will give us an image output. The need for a metric to properly evaluate our results arises. Given that our goal is to predict future page accesses, a solution is to decompose the image back to a timeseries and compare it with the ground truth values.

In the section 3.2.2 we saw that we work with images where each point of the timeseries is assigned to one pixel on the x-axis throughout multiple images, but on the y-axis, a small amount of compression has occurred, where more than one page values correspond to a specific pixel. Given that, for the image decomposition, all we have to do is to parse the images from left to right and for each column check for the pixel having a different value from the rest. We then get the y coordinate value of that pixel (counting from the bottom towards the upper part of the image) and append the column number of that image and the found value pair to a list. An example can be seen in Figure 3.10.

By creating an image from the given timeseries and then going back from that image to a timeseries we have lost accuracy on the page value because now each point corresponds to a page range. Thus we will compare our results with the "modified" ground truth, where each page value follows the same transformation we did while creating the images to find the y-axis value.

Going back to the goal of creating a metric for our models, because our timeseries has a specific page value y_t and we predict a range of page values \bar{y}_t , the range of our result may

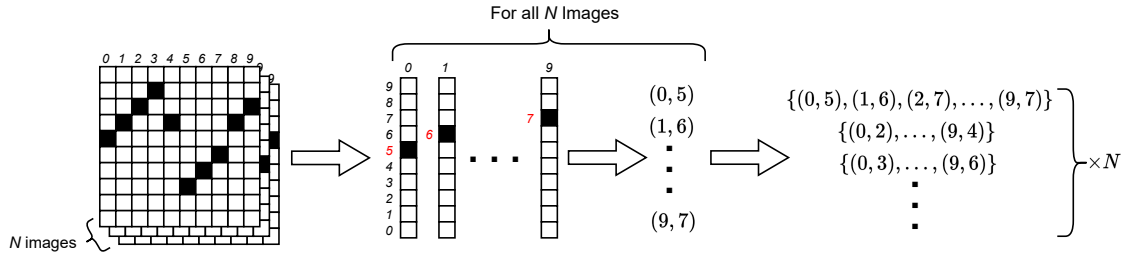


Figure 3.10: Decomposition of each image to a list of pairs of interest.

or may not contain the page y_t . This is an equivalent case of calculating the *top-k* [65] accuracy score for a classification problem, where k is equal to our δy because we consider all of the pages in the page range as the predictions with the highest score. Similar to the methodology followed in [33]

All of the comparisons and evaluations of the models we worked with will be done in a similar matter, and results will be calculated for each image output with the corresponding *modified* ground truth or by concatenating all the outputs and comparing them with the *modified* concatenated ground truth list of the equivalent images.

3.2.4 Image Dataset

To create the image datasets for each benchmark multiple images will be created to represent the entire timeseries. From the sequence $\{\bar{y}_1, \bar{y}_2, \dots, \bar{y}_t, \bar{y}_{t+1}, \dots\}$ to create an image we use as many timestamps as the x-axis size in pixels and then shift by fh timestamps to create the label for that image, where fh is the *forecasting horizon*. The label can either contain only the new future timestamps or part of the input and the future timestamps, the latter is a case of overlapping between the input and label image.

- **Input & Output image with overlap**

A good practice to implement when achievable is to try the input and the output image to have the same size. Which means that depending on the fh value we have different *overlap* ratios. The overlap ratio is defined as follows:

$$overlap = \frac{\#x\text{-axis pixels} - fh}{\#x\text{-axis pixels}}$$

Figure 3.11 shows examples of different *forecasting horizon* and thus *overlap* values.

- **Input & Output with no overlap**

Because there is no overlap, for fh values different than the x -axis size of the input image, lead to label images different in size. Figure 3.12 gives us an example of input/label images with no overlap.

- **Multiple Inputs & One Output (Many-to-one)**

Some experiments where done for models accepting more than one input images before predicting an image(many-to-one models).An example of creating such dataset can be seen on Figure 3.13.

The *step* value specifies by how many timestamps we shift with in the timeseries to get the next input image and then its label. The *step* value is the main variable that determines how many images we will have in our dataset. In most cases we will have $step = fh$, which

means all points (except from starting and ending points) will be used as an input and as a label.

In both of the above cases and throughout all of our work we do not change the size of input and label image on the y-axis.

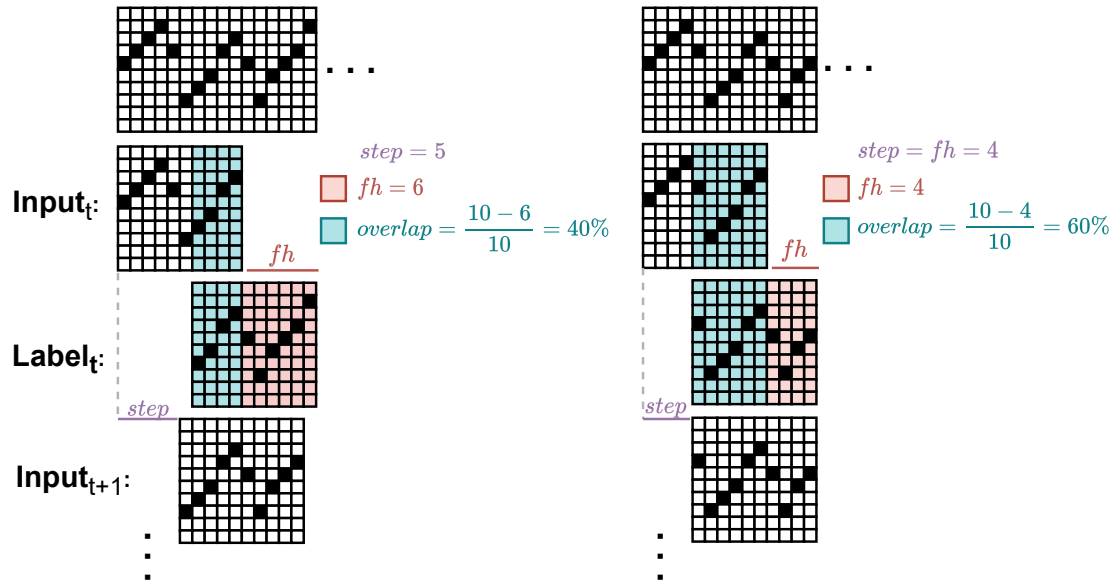


Figure 3.11: Input and label example for $fh=6/overlap=40\%$ and $fh=4/overlap=60\%$

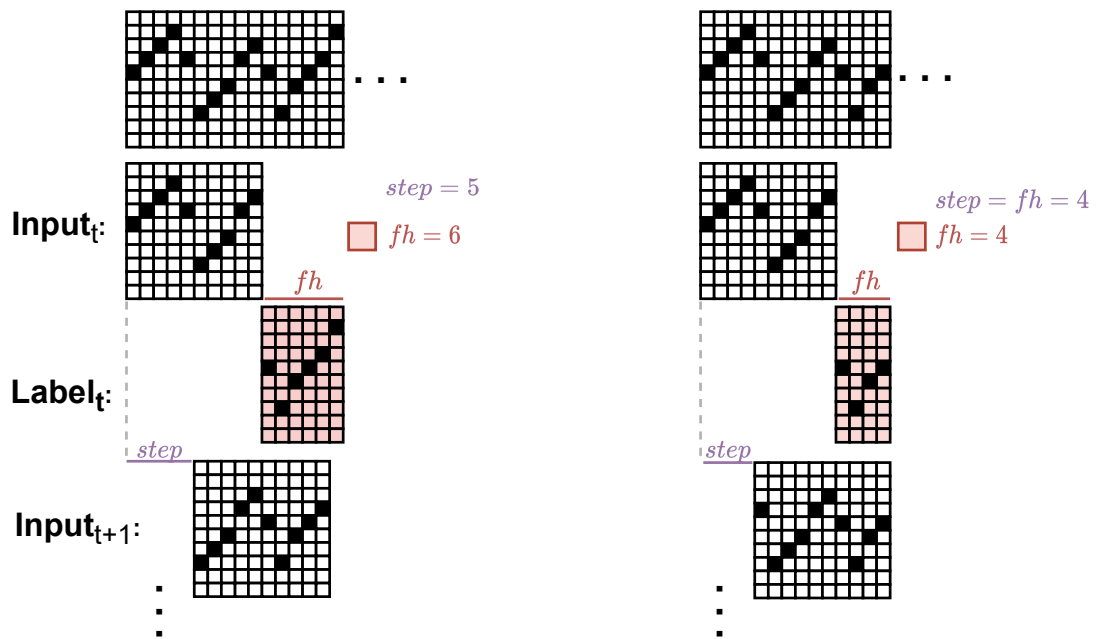


Figure 3.12: Input and label of different size example for $fh=6$ and $fh=4$

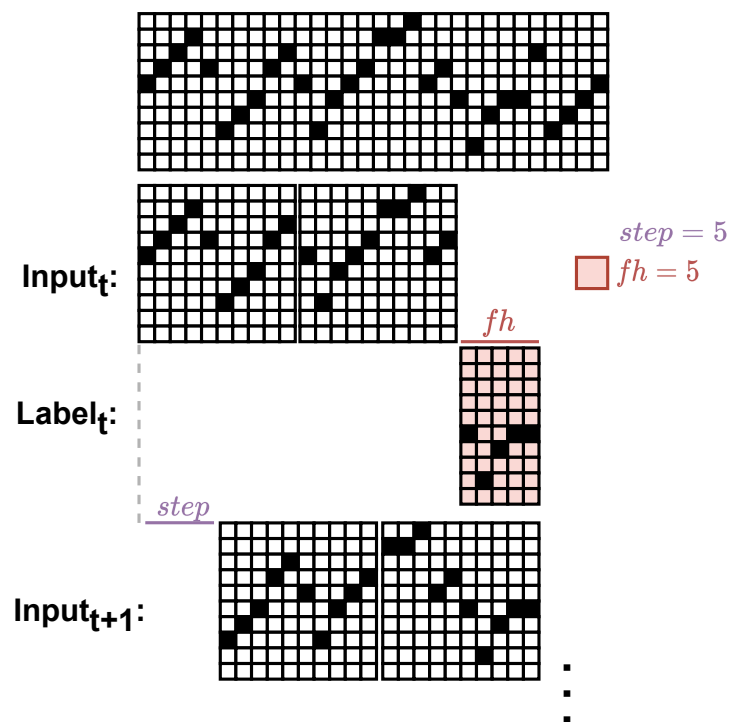


Figure 3.13: Multiple input images example

Chapter 4

Forecasting With Image-based Machine Learning Methods

On chapter 2 & 3 we presented some advantages of using image based machine learning and computer vision techniques for forecasting timeseries. Also in the last chapter we explained how from the raw given data we proceed to transform and visualize it to datasets of thousands of 2D images. Furthermore, we explained how specific design decisions affect how the input images and labels are created.

In the computer vision sector, autoencoder-based models have showed promising results for tasks such as image denoising, compression, completion and inpainting [7, 52, 28]. In this context, it is feasible to utilize autoencoder-based models, where the model's input and output are both images, or alternative models, where the input is an image but the output is numeric data. Furthermore, autoencoders have been utilized for time series forecasting with many modifications (e.g., vanilla, convolutional, recurrent, etc.) [27, 68]. Using an image-to-image method, pure computer vision models such as convolutional autoencoders or a mix of computers vision models (ex. *CNNs*) and time-series models (ex. *RNNs*) such as ConvLSTM or *LRCN* can be used. These are often utilized in next-frame video prediction tasks [1].

4.1 Experimental Methodology

When dealing with machine learning tasks having a structured experimental methodology is crucial, as it provides a systematic approach to the problem, ensuring that the outcomes are reliable. Following a structured and proper methodology means setting clear objectives, defining thoroughly our steps for data collections and preprocessing steps, and eventually documenting the model selection, training, testing and evaluation. This helps us guarantee the validity of our results and allows for a fair comparison between different algorithms or techniques. Moreover, adhering to a specified methodology helps us in identifying factors that can impact the outcome and eventually assist in creating better models. In conclusion, a clear and well documented experimental methodology is necessary for machine learning research to be reliable and trustworthy.

Bellow we specify where each part of our experimental methodology is presented within the thesis:

- Data collection and preprocessing stages are addressed in chapter 3.
- Data splitting for training, validation and testing in the sub-chapter 4.2.

- The image based Deep Learning models chosen are presented in the sub-chapter 4.3.
- The evaluation metrics utilized are analyzed thoroughly in the sub-chapter 4.4.

4.2 Data Splitting

When working with timeseries data, you need to be mindful of the temporal and spatial aspect and ensure that your data splitting methods respect the order of the data. The data splitting chosen plays a crucial role in the success of deep learning models. The robustness and generalizability of the results obtained from a deep learning problem are determined by the technique used to train, validate, and test these models.

There are several techniques used when splitting timeseries data such as rolling window cross-validation [17], expanding window cross-validation [82], walk-forward validation [56] to name a few.

Eventually our goal is to work with blocked cross-validation also known as sliding window approach 4.2. This specialized technique is employed in the field of machine learning to accurately assess the performance of a model while considering certain inherent structures or dependencies in the dataset. Unlike traditional cross-validation, where data is randomly partitioned into subsets for training and validation, blocked cross-validation involves grouping the data into distinct blocks or segments based on specific attributes or characteristics. This approach is particularly useful when dealing with data that possesses temporal, spatial, or other sequential relationships, as it ensures that such relationships are preserved during the evaluation process. Observing the "clipped" $bc - 10$ dataset as shown in image 4.1 we can see that in the first part of the dataset most strides expand within a greater Δcc but at some point and after most strides are confined within a $\Delta cc' \leq \Delta cc$. By maintaining the integrity of these blocks, blocked cross-validation provides a more realistic representation of a model's generalization performance, offering insights that are better aligned with real-world scenarios where data dependencies play a critical role. Another reason for preferring this method is for reducing the training and testing time compared to other methods such as expanding window cross validation etc.

Unfortunately given how many models we needed to train and experiment with, employing such method as blocked cross-validation would be extremely time-consuming. On Chapter 5.6 we will touch upon this subject again and eventually experiment with training the model throughout the dataset. All other testing from here after is based on the idea (seen in [57, 77] that we train on the first part of the dataset, specifically 10 – 30%, and then test for the rest of it. The first 10% is omitted because ([77], we observed that during that part of the dataset the trace is a lot more scarce and eventually led to worse models what it was included. Given the above, it is obvious that we are creating a hard problem to solve because as already noted the dataset seems to have a different behavior in different parts of the dataset. Also unless specified otherwise all testing was done on the bc benchmark.

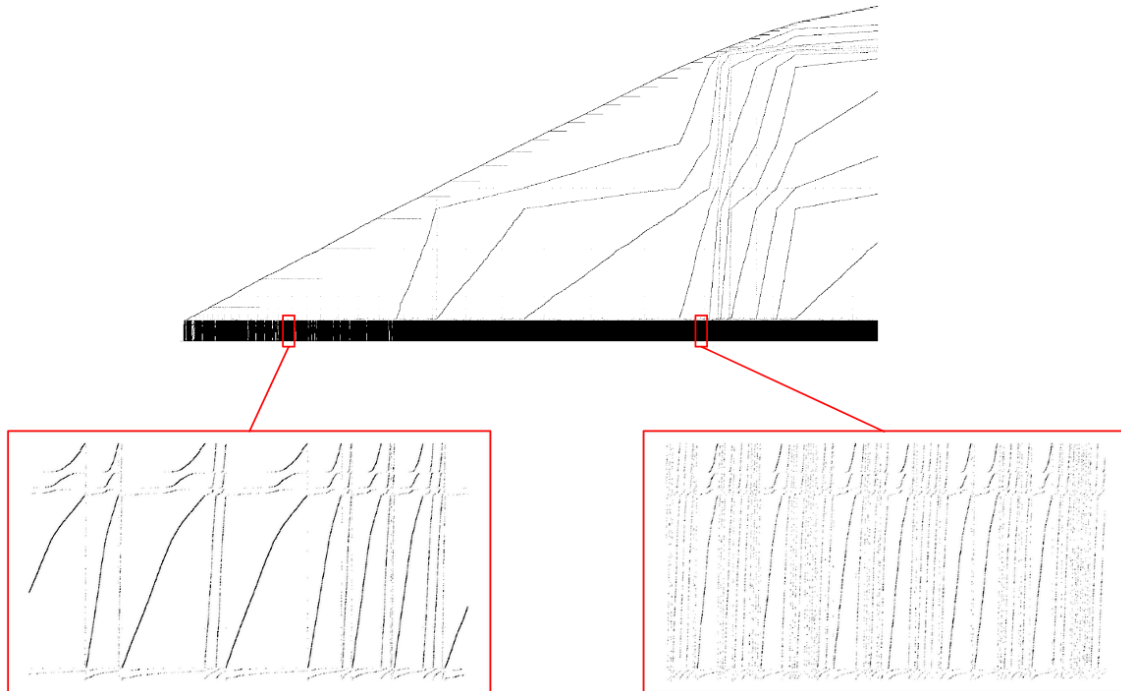


Figure 4.1: Visualized difference of "density" of stride-like patterns within the same benchmark. In this visualization the Δ_{cc} is the same between the two zoomed-in images (images within the red bounding boxes)

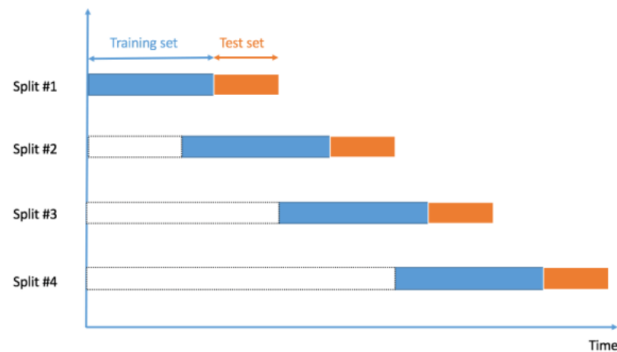


Figure 4.2: Sliding window cross validation example. Image Source: www.r-bloggers.com

4.3 Deployment of the Image Based Deep Learning Models

Throughout the entire research of the thesis, multiple models have been tested. For example, as briefly described in 3.2.1, a significant amount of time and effort was invested in finding suitable forecasting models for a dataset of images created using a different approach than the one we eventually adopted deeming all those models developed useless. To maintain the thesis's scope and coherence, we have chosen not to include the work and findings related to models that do not align with the final visualization we adopted.

Furthermore, pretrained Convolutional Neural Networks (*CNNs*) models were considered for our task. Pretrained models generally deliver impressive results across various tasks, their effectiveness though is limited in our scenario where the input data exhibits a highly

specific structure (binary images, one "value" per column etc.). These models derive their powerful capabilities from being trained on extensive, diverse and in most cases real world datasets, enabling them to learn complex patterns and features in a generalized manner. However, when confronted with our generated images that possess unique and specialized structures, these pretrained models encounter challenges. This is primarily due to their inherent biases towards statistical regularities found in conventional real-world images, which may hinder their ability to comprehend and capture the distinct complexities presented in the generated images.

The current *state-of-the-art next frame prediction* models do not suit our specific case due to their limitations in handling our unique requirements. Many of these models are designed for real-world images and videos, which may not align with the characteristics of our data. Even in cases where generated data, such as the *Moving MNIST* dataset [79], is used, the models typically focus on predicting the next few frames with small pixel movements. Generally these models predict the "movement" of an object and not expected future values. Our goal necessitates predicting frames much further into the future, making it a different problem .

Consequently, achieving optimal performance for our task involving such generated images necessitates experimenting with models from scratch and training exclusively on our dataset. Only then can the models be adequately adapted to the specific structural characteristics of our generated data and exhibit the desired performance for our task.

Eventually we ended with two model approaches, both of them consisting of ConvLSTM layers (explained in chapter 2 and [74]). The differentiating factor being the input image or images and the corresponding output image dimensions.

ConvLSTM-Mult-Img

Our model follows an autoencoder like structure, where at first the image is downsampled similarly to the encoder network and then the image is up-sampled again to the starting input dimensions, similar to a decoder network. Moreover in the specific model for the encoder phase we have a *ConvLSTM* layer followed by a batch normalization layer, then a dropout layer and finally an down-sampling layer. The same structure is followed for the decoder with the only difference that we start with an up-sampling layer and then proceed with the *ConvLSTM*, batch normalization and dropout layers. Additionally due to the vanishing gradient problem, where gradients become very small as they propagate backward through numerous layers we have added a residual connection to help with our model training and performance. The detailed architecture of the model developed in *TensorFlow* can be examined in 4.3. The model above can have as input one or more images of $H \times W \times C$ (where $C = 1$ because the images in our datasets have only one channel) dimensions and the output will be one image of the same $H \times W \times C$ dimensions, where the W corresponds to the forecasting horizon fh length.

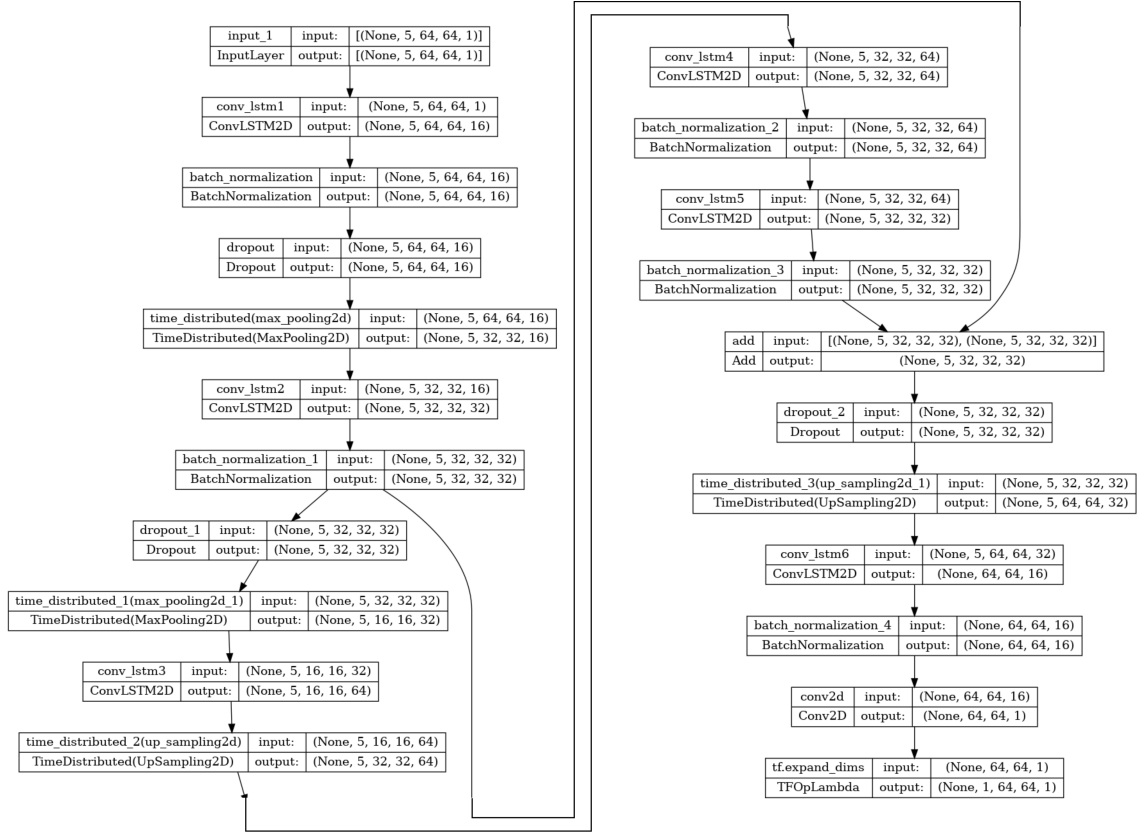


Figure 4.3: *ConvLSTM-Mult-Img* Model Architecture. The values and dimensions appeared are from a random example as they are dependent from the size of the input image.

ConvLSTM-One-Img

This model follows the same structure as with the previous one with the main difference being that after down-sampling the input image we flatten the output, forward it to two dense layers and eventually reshape the the vector before we continue with the up-sampling. Applying the proper number of units for the dense layers and the reshaping parameters we can define the dimensions of the final output image. In this model we do not have a residual connection. The detailed architecture of the model developed in TensorFlow can be examined in 4.4. This model accepts one input image of $H \times W \times C$ (where $C = 1$ because the images in our datasets have only one channel) dimensions, and the output is one image of $H \times W' \times C$, where W' corresponds to the forecasting horizon fh length.

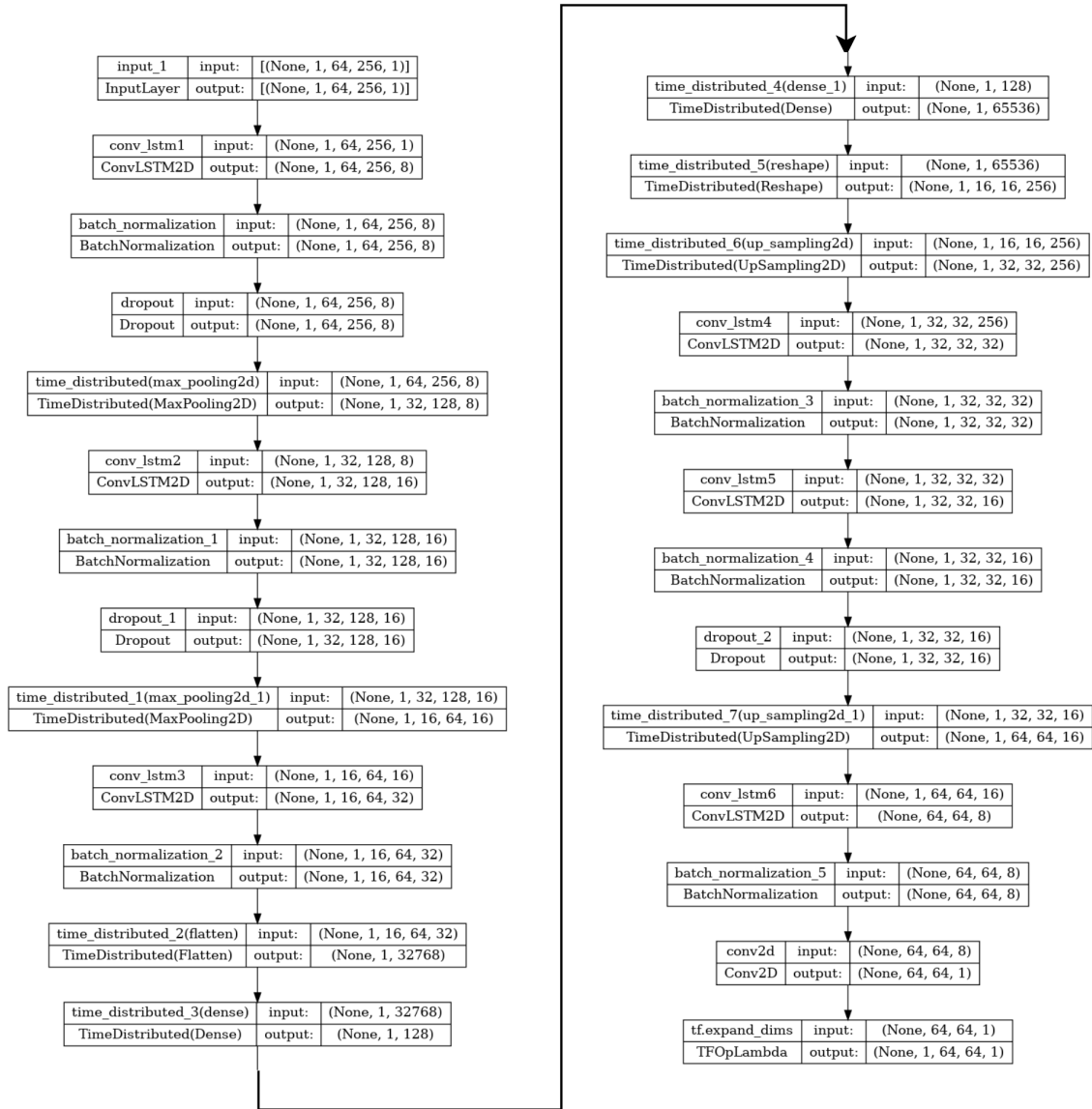


Figure 4.4: *ConvLSTM-One-Img* Model Architecture. The values and dimensions appeared are from a random example as they are dependent from the size of the input image.

Hyperparameters

In both of the aforementioned models, the same hyperparameters were used during the training process. The selected optimizer is Adam [46], and the initial learning rate is set to 0.001. Reduction of the learning rate on a plateau is implemented, wherein if the validation loss does not decrease for 10 consecutive epochs, the learning rate is reduced by a factor of 10. To prevent overfitting and minimize unnecessary training time, early stopping is also employed. Specifically, if the validation loss does not decrease for 15 consecutive epochs, the model halts training. Lastly, the best model from the training process is saved in the .hdf5 format.

As for the loss function *Mean Squared Error* was found to work better for both the *ConvLSTM-Mult-Img* and *ConvLSTM-One-Img* models. *Binary-Cross-Entropy* [69] yielded better results in very few cases and also led to slightly slower training times. In future

studies, a more thorough analysis could be conducted, potentially exploring the development of a custom loss function to better accommodate the unique characteristics of our images, such as their binary nature with only one value per column.

Output Binarization

The output of the above models is a 2D image (2D array). Given that our models are *CNNs* with *ReLU* activation the output images will have values between zero and one. This represents a gray-scale intensity level. For our case we are interested in the most "confident" prediction for each column. Thus we apply a "Binarization" where the brightest (whitest from the gray-scale image) pixel of each column, is the one kept and rounded up to a value 1.0, effectively creating a binary image. An example can be seen in Figure 4.5.

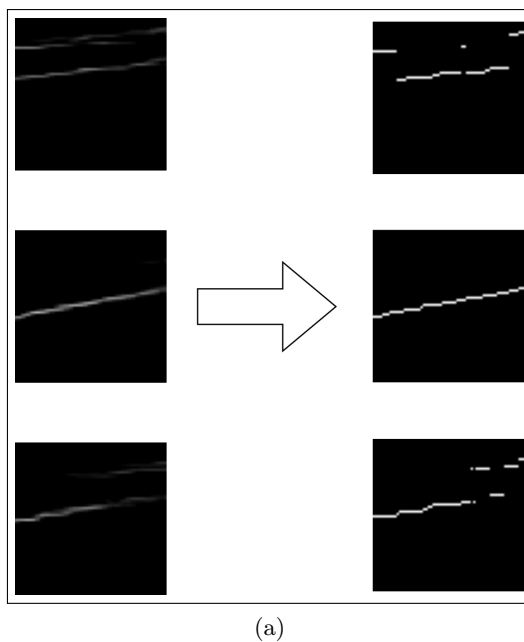


Figure 4.5: Prediction Images before (left) and after (right) they are "Binarized". The brightest pixel of each column is kept and rounded to a value of 1.

4.4 Evaluation Metrics

Choosing appropriate metrics for a given task can be challenging, as different metrics capture different aspects of performance and may prioritize different types of errors. In this section, we provide an overview of the evaluation metrics used in timeseries forecasting. Moreover, recent work [19], highlight the importance of using image-based evaluation metrics, because common numeric metrics may not capture all the information concerning timeseries forecasting problems properly and could give misleading and poor results. We thus discuss their strengths and limitations and provide our reasoning for choosing each metric.

Evaluation metrics

For the numeric metrics presented below when we are referring to the forecast value \hat{y}_i and the corresponding observed ground truth y_i we are using the results of the predicted image decomposition explained in the subsection 3.2.3.

Accuracy - Dice coefficient

The *Dice coefficient* is a statistical measure commonly used in image segmentation and pattern recognition tasks to evaluate the similarity between two sets of data. It is computed as follows:

$$Dice = \frac{2 \cdot |Y \cap \hat{Y}|}{|Y| + |\hat{Y}|} \quad (4.1)$$

where \hat{Y} is the predicted image, Y the corresponding ground truth image and $|Y|$ is the cardinality of the set, which means the number of elements in the set (respectively for $|\hat{Y}|$ and $|Y \cap \hat{Y}|$).

Dice coefficient-Accuracy quantifies the overlap by calculating the ratio of twice the intersection of the sets to the sum of their sizes. A *Dice coefficient* value of 1 indicates a perfect match or complete similarity, while a value of 0 signifies no overlap. By examining the *Dice Coefficient* further, because of the unique structure of our images, binary and with only one different value pixel per column, we have that $|Y|$ and $|\hat{Y}|$ give us the forecasting horizon length. On the numerator of our equation we have that $|Y \cap \hat{Y}|$ equals to the number of right predictions because they are the only pixels having the same non-zero value between the two images/sets. Thus by taking the above observations we have:

$$Dice\ Coefficient = \frac{2 \cdot |Y \cap \hat{Y}|}{|Y| + |\hat{Y}|} = \frac{2 \cdot \#correct\ predictions}{2 \cdot \#total\ predictions} = \frac{\#correct\ predictions}{\#total\ predictions} \quad (4.2)$$

We can easily notice now that the final equation in 4.2 is the same with **Accuracy**.

MAE_{scaled}

MAE_{scaled} is based on the *Mean Absolute Error* metric (MAE) is an evaluation metric to measure the difference between two vectors. It is computed as follows:

$$MAE_{scaled} = \frac{\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|}{y\text{-axis pixels}} \quad (4.3)$$

where \hat{y}_i is the forecast, y_i the corresponding observed ground truth, n is the length of the time series and y-axis pixels the size of the predicted image on the y-axis/the possible classes for prediction.

It measures the average absolute difference between the predicted values and the actual values of the target variable, by summing the absolute values of their differences across all data points and dividing by the count of data points. After that we divide by the size of the prediction in the y-axis, thus the possible predictions in order to scale our results compared to the granularity of the image. Unlike MAE this metric is not scale independent, because of the included division by the y-axis size. For example if the average distance of the predicted value from the ground truth is 5, this result can be consider very bad if our y-axis size is 16, but could be considered very good if our y-axis size is 128. The MAE_{scaled} distance provides a measure of how far the mean predicted values are from the actual values scaled to the image precision on the page-range. As it can be understand a lower

MAE_{scaled} distance value indicates that the model's predictions are more accurate, while a higher mae_{scaled} distance value indicates that the model's predictions are less accurate.

Root Mean Squared Error

Root Mean Squared Error (RMSE) is an evaluation metric commonly used to measure the accuracy of a regression model's predictions. It is computed as follows:


$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (4.4)$$

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

The above equation measures the difference between the predicted values and the actual values of the forecasted variable, by taking the square root of the average squared difference between them. The RMSE provides a measure of how far the predicted values are from the actual values, in the same units as the target variable. A lower RMSE value indicates that the model's predictions are more accurate, while a higher RMSE value indicates that the model's predictions are less accurate.

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 [25, 67]. We extend the metric to measure the forecast accuracy, similarly to the work done in [19]. It is computed as follows:

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (4.5)$$


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. 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.

Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) expresses the average absolute percentage error across all data points and is a widely used evaluation metric for assessing the accuracy of forecasting or prediction models. It is calculated as follows:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \cdot 100 \right| \quad (4.6)$$

where \hat{y}_i is the forecasted value, y_i is the corresponding observed (actual) ground truth value, and n is the total number of data points (samples) in the time series.

Unlike $RMSE$, $MAPE$ is a scale independent metric since it is calculated as a percentage. $MAPE$ is valuable for comparing the performance of different forecasting models or prediction techniques, particularly when dealing with datasets of varying scales. Like any metric, $MAPE$ also has some limitations. First of all, issues can occur when the actual values are close to zero, as it could lead to undefined or infinite values. Furthermore, it treats all data points equally, regardless of their significance. Additionally, $MAPE$ tends to heavily penalize large errors due to its absolute percentage nature. In summary, $MAPE$ is a useful metric for quantifying the accuracy of forecasting models, especially in scenarios where percentage errors are more informative or meaningful than raw differences.

SSIM

The *Structural Similarity Index (SSIM)* is a metric used to assess the similarity between two images. It is computed as follows:

$$SSIM = \frac{(2\mu_{\hat{y}}\mu_y + c_1)(2\sigma_{\hat{y}y} + c_2)}{(\mu_{\hat{y}}^2 + \mu_y^2 + c_1)(\sigma_{\hat{y}}^2 + \sigma_y^2 + c_2)} \quad (4.7)$$

where:

- y and \hat{y} are the compared images,
- $\mu_{\hat{y}}$ and μ_y are the mean values of \hat{y} and y respectively,
- $\sigma_{\hat{y}}$ and σ_y are the standard deviations of \hat{y} and y respectively,
- $\sigma_{\hat{y}y}$ is the cross-covariance between \hat{y} and y
- c_1 and c_2 are small constants added for numerical stability.

It measures the perceived quality of an image by considering three components: luminance, contrast, and structure. $SSIM$ compares the local patterns and structures of pixels in the predicted and ground truth images. It computes a similarity index that ranges from -1 to 1, with 1 indicating a perfect match. The $SSIM$ index takes into account the mean values, standard deviations, and cross-covariance of the compared images. It captures both the differences in pixel intensities and the variations in structural information.

Inference Time

Inference time is the time it takes for the model to make a prediction given an input. This metric turned out not to be that straightforward as multiple things could alter its results drastically. To be more precise, the total inputs and outputs provided during testing by a model is highly dependent on the *forecasting history* and *forecasting horizon*. For example a model prediction length of 256 will test in total significantly less samples compared to a model with a forecasting horizon of 16 (around $16x$ times less inputs). Thus in order to be fair we need to compare inference time per prediction. For calculating the later used two solutions. The first is by giving the model each time one input and calculating the time for the output to be computed, we do this for 5000 single inputs and calculate the mean. The second way, is by giving all 5000 images, a batch size of 1, and calculating the time needed to get all 5000 outputs. Even though the first method seems to be the most objective, probably due processes running on the background, when starting the prediction each time, almost all results (for our model) have a fixed value somewhere in between $25 - 35ms$, even if the model is a lot more complex (thousands more parameters). On the other hand the second method captures the slight difference in computation time for the more complex models and shows an increase as the parameters increase. Thus in all tables both metrics will be included in the last column.

Inference time refers to how long a model takes to generate a prediction after receiving new data. However, measuring this metric can be tricky because several factors can significantly impact the results. To be more precise, the total number of inputs and outputs provided during testing by a model is highly dependent on the *forecasting history* and *forecasting horizon*. For example, a model predicting 256 steps into the future will process considerably fewer samples compared to one predicting only 16 steps ahead (roughly $16x$ times less). Given the nature of the problem we try to solve and to ensure a fair comparison, we need to consider inference time per prediction. Here, we explore two methods for calculating this: **(a)** Single Input Method: The model receives one input at a time, and the time taken to generate the corresponding output is measured. This is repeated for 5000 individual inputs, and the average time is calculated. **(b)** Batch Input Method: All 5000 inputs are provided to the model at once, but it is given a batch size of 1. The total time taken to generate all 5000 outputs is measured. While the single input method might seem more objective, it often produces a fixed value (around $25 - 35$ milliseconds) for most models, even complex ones with many more parameters. This is likely due to background processes influencing the initial setup of the prediction. On the other hand, the batch processing method captures the subtle variations in computation time for more complex models. It reveals a gradual increase in processing time as the number of parameters grows. Therefore, both metrics (single input and batch input) will be included in tables to provide a more comprehensive picture of inference time for forecasting models.

Section Takeaway:

After presenting all performance metrics that will be utilized in our analysis further on, it must be noted that the main metric we will evaluate upon is the **Accuracy-Dice Coefficient**. For simplicity we will refer to this metric simply as **Accuracy**. To be more specific, all other metrics provide us with information on how "close" our results are to the *ground truth* values, and will be used accordingly. But, the only metric that will determine the performance of our models in a real world application is **Accuracy**, as nearby values, even thought may provide low errors, eventually lead to wrong prediction and thus a wrong prefetch.

4.5 Tuning of the Forecasting History

Forecasting history refers to past observations and data points in a timeseries dataset. In our case it effects the number of timestamps present in our generated input images. Within the forecasting history, patterns, trends and other relevant information is embedded from previous time intervals. The importance of forecasting history in timeseries prediction lies in its ability to provide essential context and insights into the behavior of the variable over time. The historical context enables the model to make more accurate forecasts and reliable predictions of future values. The goal of this analysis is to observe how each of our models reacts when we are changing the forecasting history. A lot of experiments were done for various Δcc sizes, but given that this approach was abandoned the results found are dismissed.

In this analysis we proceed by presenting how forecasting history effect the performance of our models. In both cases the forecasting horizon (prediction) is fixed to a value of 64 timestamps and similarly the vertical dimension (y-axis resolution) is set to 64 pixels.

Having established a predictive horizon of 64 timestamps into the future, we have to configure the *ConvLSTM-Mult-Img* model accordingly. This configuration mandates an input image size of 64 pixels along the x-axis. In order to facilitate a comparative analysis of our outcomes against those of our other model, we engaged in experiments with different number of input images, as seen in Table 4.1. The table presents our experimental setup, which includes 1, 2, 3, 4, 5, and 8 images, respectively. Within the first column of the table, the number of total timestamps given as forecasting history is indicated within the parentheses.

In the case of the *ConvLSTM-One-Img* model the forecasting history can be any size of choice. For our case, as seen in Table 4.2 we have tested for forecasting history values of 16, 32, 64, 128, 256, 512 and 768 timestamps. Thus creating input images of a corresponding x-axis size.

Upon reviewing Table 4.1 and figures in 4.6, it becomes apparent that augmenting the input image count to two, thereby extending the predictive history in terms of samples, results in a noteworthy and significant enhancement across all performance metrics. Subsequently increasing the input images to three yields a marginal yet evident improvement in the performance metrics. However, pushing the total input image count beyond three shows a plateau of improvement on our metrics. In contrast, providing even more images as input, leads to deterioration in our outcomes. It is also important to note that even from an input of 1 image, the inference time is greater than that of *ConvLSTM-One-Img*, and by increasing the input images the inference time increases significantly.

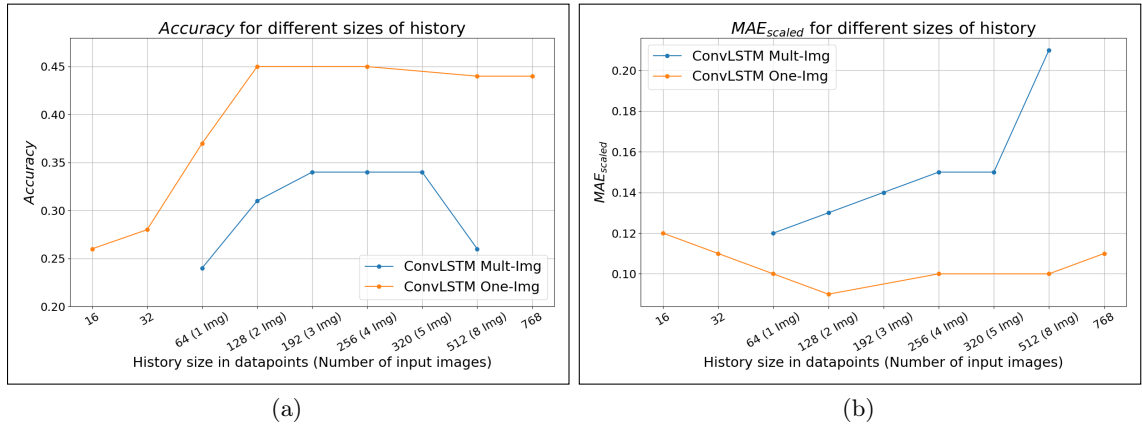
By analyzing Table 4.2 and figures in 4.6, it is evident that increasing the forecasting history size leads to improvements across all performance metrics. Upon closer examination, a significant improvement is noted when the forecasting history is increased to match the forecasting horizon. Beyond this point, further increase of the forecasting history continues to yield better results, but, the increase isn't as intense it was before. It was clear from additional testing with forecasting histories greater than 512 that all measures started to decline. An interesting observation is how the inference times does not change with the increase of the input size, showing the grate capabilities of *CNNs* in parallelized computation.

input images (data points)	Accuracy	MAE _{scaled}	RMSE	IoU	MAPE	Inference Time [ms]
1 (64)	0.24	0.12	11.73	0.17	0.92	43.35/5.35
2 (128)	0.31	0.13	12.40	0.21	1.00	49.22/9.58
3 (192)	0.34	0.14	13.17	0.25	0.75	50.79/12.17
4 (256)	0.34	0.15	13.19	0.25	0.66	57.77/16.69
5 (320)	0.34	0.15	13.23	0.25	0.60	59.53/17.63
8 (512)	0.26	0.21	17.70	0.19	1.43	69.79/30.71

Table 4.1: Performance metrics of *ConvLSTM-Mult-Img* model for different numbers of input images ($64 \times 64 \times 1$) and $fh = 64$

input (X')	Accuracy	MAE _{scaled}	RMSE	IoU	MAPE	Inference Time [ms]
16	0.26	0.12	11.00	0.18	0.89	43.68/5.41
32	0.28	0.11	10.20	0.18	0.65	46.36/5.23
64	0.37	0.10	10.01	0.26	0.57	45.12/5.42
128	0.45	0.09	9.82	0.33	0.55	44.68/5.37
256	0.45	0.10	10.22	0.33	0.58	45.25/5.85
512	0.44	0.10	9.98	0.33	0.51	44.26/5.48
768	0.44	0.11	11.38	0.32	0.62	46.12/5.63

Table 4.2: Performance metrics of *ConvLSTM-One-Img* model for input images of different x-axis size ($64 \times X' \times 1$), as forecasting history changes for the *ConvLSTM-One-Img* model for $fh = 64$



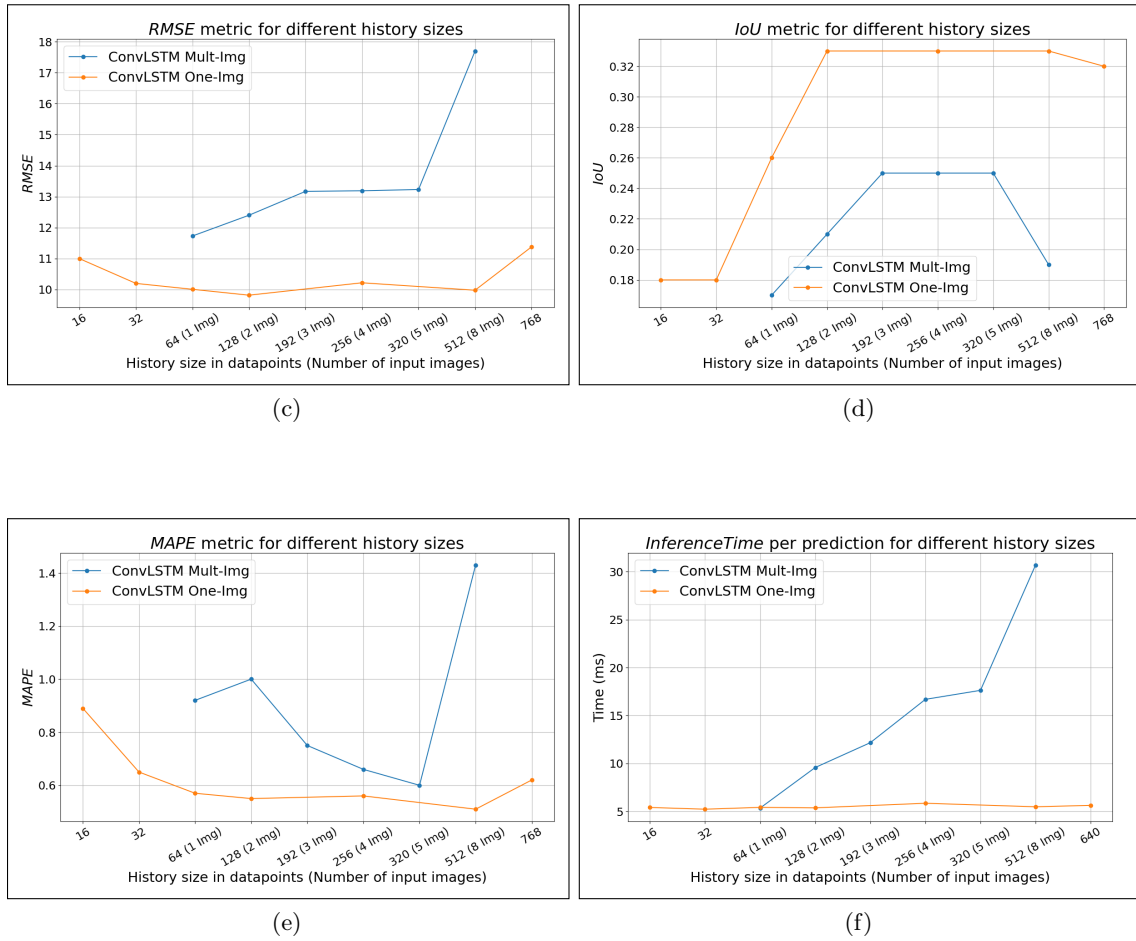


Figure 4.6: Plots using matplotlib for metrics results shown on Tables 4.1 and 4.2: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) $RMSE$ (d) IoU (e) $MAPE$ (f) $Inference\ Time$ for 1 prediction.

Section Takeaway:

After examining the performance metrics for the *ConvLSTM-Mult-Img* and *ConvLSTM-One-Img* models for the various given input sizes, it is obvious that the **later model** performs significantly better in all scenarios and shall be the model that we will continue further on in our analysis. The *ConvLSTM-Mult-Img* not only struggles to perform adequately, but also, has an increasing Inference Time which is not accepted in such a time sensitive scenario. Furthermore, taking into consideration at first the **Accuracy** and then the rest metrics on Table 4.2 we will be proceeding with either a **Forecasting History** of **128** or **256**.

4.6 Tuning of the Image Height Size

In chapter 3.2.2 we explained how the visualization chosen is "lossy" due to the compression that happens where more than one pages can correspond to the same y-axis pixel. Depending on the y-axis pixel size of the image, each pixel of the visualization associates to a specific page range. Choosing the right value is not a trivial task and neither is a clear answer on which is the most suitable value. In Table 4.3 and 4.4 we examine this "page granularity"

for various different sizes and evaluate based on our performance metrics. The former table is for forecasting history value of 128 and the latter for a forecasting history value of 256.

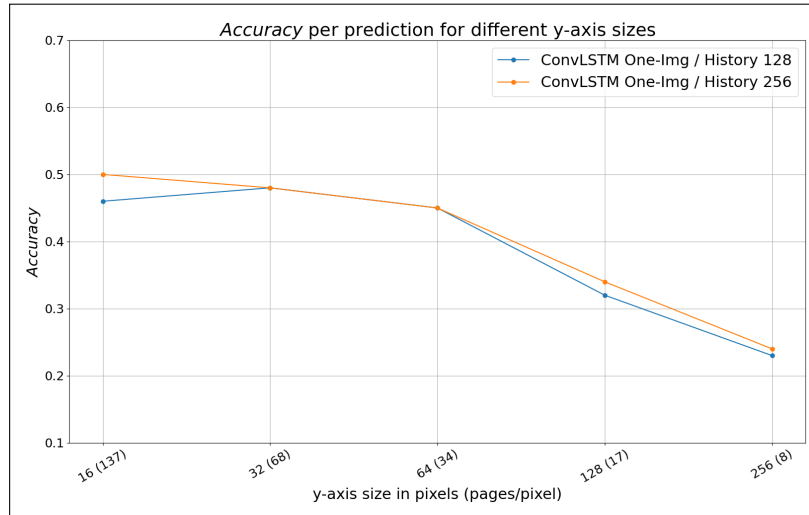
y-axis size Y' (pages/pixel)	Accuracy	MAE_{scaled}	RMSE	IoU	MAPE	Inference Time [ms]
16 (137)	0.46	0.03	02.97	0.35	0.39	44.6/5.47
32 (68)	0.48	0.05	05.07	0.36	0.49	43.81/5.37
64 (34)	0.45	0.09	9.82	0.33	0.55	44.67/6.37
128 (17)	0.32	0.18	19.37	0.22	0.55	44.85/5.43
256 (8)	0.23	0.37	39.92	0.15	0.62	46.67/5.83

Table 4.3: Performance metrics of *ConvLSTM-One-Img* model for input image of ($Y' \times 128 \times 1$) and $fh = 64$

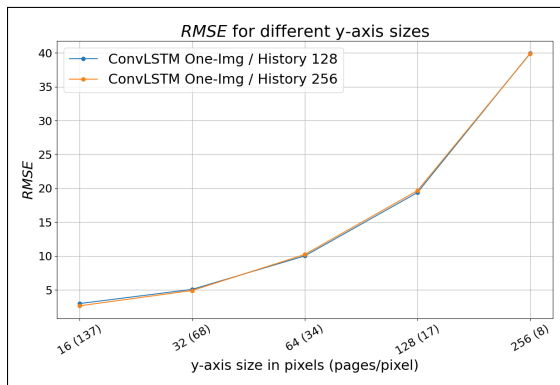
y-axis size Y' (pages/pixel)	Accuracy	MAE_{scaled}	RMSE	IoU	MAPE	Inference Time [ms]
16 (137)	0.50	0.03	02.64	0.38	0.42	44.1/5.51
32 (68)	0.48	0.05	04.90	0.36	0.45	44.16/5.43
64 (34)	0.45	0.10	10.22	0.33	0.58	45.25/5.84
128 (17)	0.34	0.18	19.65	0.24	0.59	44.68/5.70
256 (8)	0.24	0.37	39.87	0.16	0.61	46.94/6.28

Table 4.4: Performance metrics of *ConvLSTM-One-Img* model for input image of ($Y' \times 256 \times 1$) and $fh = 64$

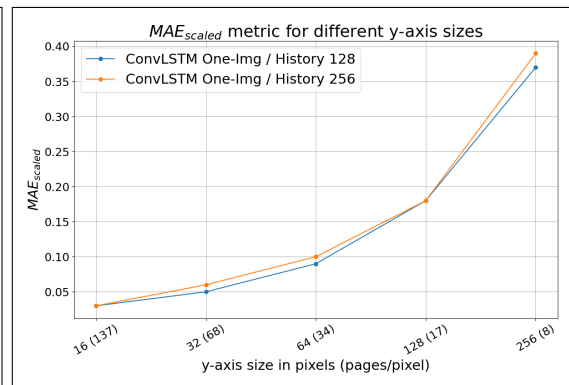
Examining our metrics it is not straightforward how to evaluate the results. First of all, based on the definition of MAE_{scaled} in 4.4 the value of the y-axis dimension directly affects the metrics value. For example, a scenario with a high vertical dimension, thus more classes and a greater granularity, a bigger absolute difference can provide the same result for the MAE_{scaled} metric when compared to a case with a smaller absolute difference, but a smaller number of values on the y-axis. The number of possible values on the y-axis, even for metrics seemingly independent of it, ultimately affects the prediction’s difficulty and accuracy. With a larger range of possible y-axis values, predictions become more complex due to the increased number of possibilities. This complexity translates to a higher chance of errors. On the other hand, a limited range of y-axis values simplifies the prediction process. Statistically, this leads to a higher chance of correct predictions due to fewer options. Additionally, the potential error between prediction and actual value (ground truth) is limited by the smaller range of possible outcomes. While the previous highlights the relationship between y-axis size and prediction performance, a definitive value is hard to be established. Referring to Tables 4.3 and 4.4 (along with their corresponding plots in Figure 4.7), we need to choose a suitable y-axis size to proceed. We select a value of 64 because it appears to strike a balance between two key aspects: the number of pages displayed per y-axis pixel and the performance of specific metrics, particularly accuracy. For a greater size in the vertical dimension the metric results start to deteriorate significantly, while smaller values see improvement but not that significant in order to justify the great loss of precision that has occurred while grouping the pages to classes. Similar need for grouping can be found in work done by Milad Hashemi et al. [33].



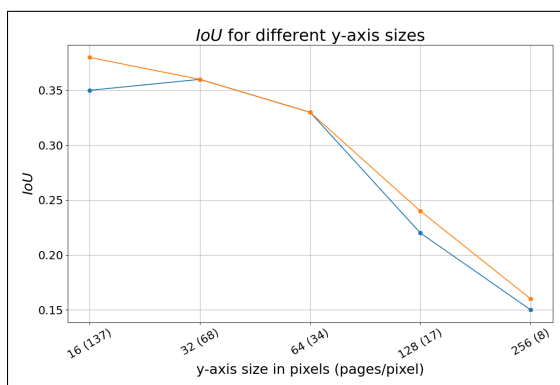
(a)



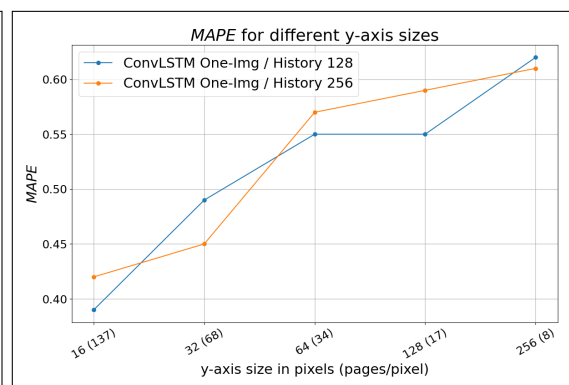
(b)



(c)



(d)



(e)

Figure 4.7: Plots using matplotlib for metrics results shown on Tables 4.3 and 4.4: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) RMSE (d) IoU (e) MAPE.

Section Takeaway:

In this section, we discussed how there is no determining factor for choosing the proper

y-axis size. While larger values minimize data compression and information loss within grouping created, but, create an increasingly harder problem to solve and compromising Accuracy. We eventually select a **y-axis** size value of **64** as we believe a middle ground is achieved between the number of pages represented by each pixel and the performance in Accuracy. A larger y-axis size, even though preferable, leads to substantially worse results in all performance metrics.

4.7 Tuning of the Forecasting Horizon

The forecasting horizon refers to the time period into the future for which predictions are made in a time series analysis. Selecting the appropriate forecasting horizon is vital as it directly impacts the accuracy and reliability of the predictions. Similarly with the previous fields, tuning the forecasting horizon directly affects the output image size and thus the complexity of the model and also determines the shift between the training samples. Hence, when the forecasting horizon is larger, the number of training samples decreases. Selecting the right forecasting horizon is again, not a trivial task, and requires a trade-off between Accuracy and the practicality of our model and plays a significant role in determining the overall usefulness and effectiveness of time series forecasting models.

The work conducted in this thesis was aimed towards a longer forecasting horizon. In almost all related work (for example [75, 57, 33]), as mentioned in 1.2, the research conducted is for prefetching 1 value in the future, thus a forecasting horizon of 1. It's crucial to understand that short-term and long-term forecasting can be seen as distinct problems. Different models perform better at different time horizons. For instance, when predicting just the very next data point (short-term forecasting), simpler models can often be surprisingly effective. These simpler models often are the ones already in use and are generally easier to implement on hardware, making them a practical choice for short-term forecasting at this time of writing. Of course the above, does not mean that sophisticated and complex solutions such as the ones seen at [75, 57, 33, 89, 90] do not exist. These advanced approaches often demonstrate significant performance improvements compared to traditional methods, even though none of them are used in real-world systems.

As the forecasting horizon increases, especially to significantly longer periods like 16 or more timestamps, the problem becomes fundamentally different. We shouldn't expect similar results as the ones we can get from short-term predicting models. The work of this thesis, is to acknowledge the trade-offs involved and research and experiment upon models suited for longer forecasting horizons.

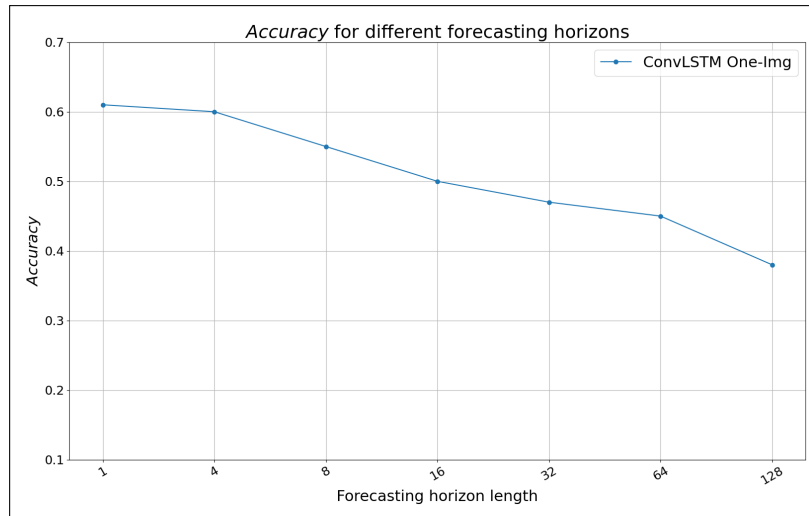
For the results presented on Table 4.5 and the corresponding plots found at Figure 4.8, we have assumed a *y-axis size* of 64 as explained in the previous sub-chapter, and a *forecasting history* of 128. While results for a *forecasting history* of 256 timestamps showed similar trends with slight metric differences, they have been omitted to avoid cluttering the presentation of the findings with excessive graphs and information, as including them would be repetitive.

Upon examination of the results, we can clearly observe how as we increase the forecasting horizon value all metrics start to deteriorate. Interestingly, the MAE_{scaled} metric remains stable across all different forecasting horizons. This suggests that while accuracy declines with longer predictions (as seen on the first column), the average absolute difference between our forecasts and the actual values doesn't increase significantly. This implies that our predictions might capture the general trend but lack precision, possibly hovering around the true values. Similarly the $RMSE$ metric has low values, growing as the forecasting horizon increases. Surprisingly, the IoU metric follows the general trend of metrics, even for the small

Forecasting Horizon	Accuracy	MAE _{scaled}	RMSE	IoU	MAPE	Inference Time pred [ms]
1	0.61	0.07	04.83	0.61	0.54	44.32/5.15
4	0.60	0.07	06.31	0.51	0.45	45.00/5.20
8	0.55	0.07	07.01	0.45	0.49	44.59/5.5
16	0.50	0.08	08.19	0.38	0.47	43.65/5.42
32	0.48	0.09	08.97	0.35	0.51	44.48/5.59
64	0.45	0.09	09.82	0.33	0.55	44.12/5.31
128	0.38	0.13	12.84	0.26	0.81	44.9/5.25

Table 4.5: Performance metrics of *ConvLSTM-One-Img* model for input image of $(64 \times 128 \times 1)$ and different forecasting horizons

values of 1 and 4, given that it is an image based metric and in the aforementioned cases, the images compared have both only a very small x-axis dimension (1 and 4 correspondingly). The *MAPE* metric deviates from the expected pattern, showing an increase in error for shorter predictions. While the exact reason is unclear, it could be due to a higher sensitivity to outliers in smaller datasets, as mentioned when introducing the metric. While accuracy decreases the further we forecast, it's crucial to consider the information gain from extending the prediction horizon. For instance, even though a 14% decrease in accuracy compared to the 1- and 32-step forecasts is significant, we gain valuable insights from 31 additional data points.



(a)

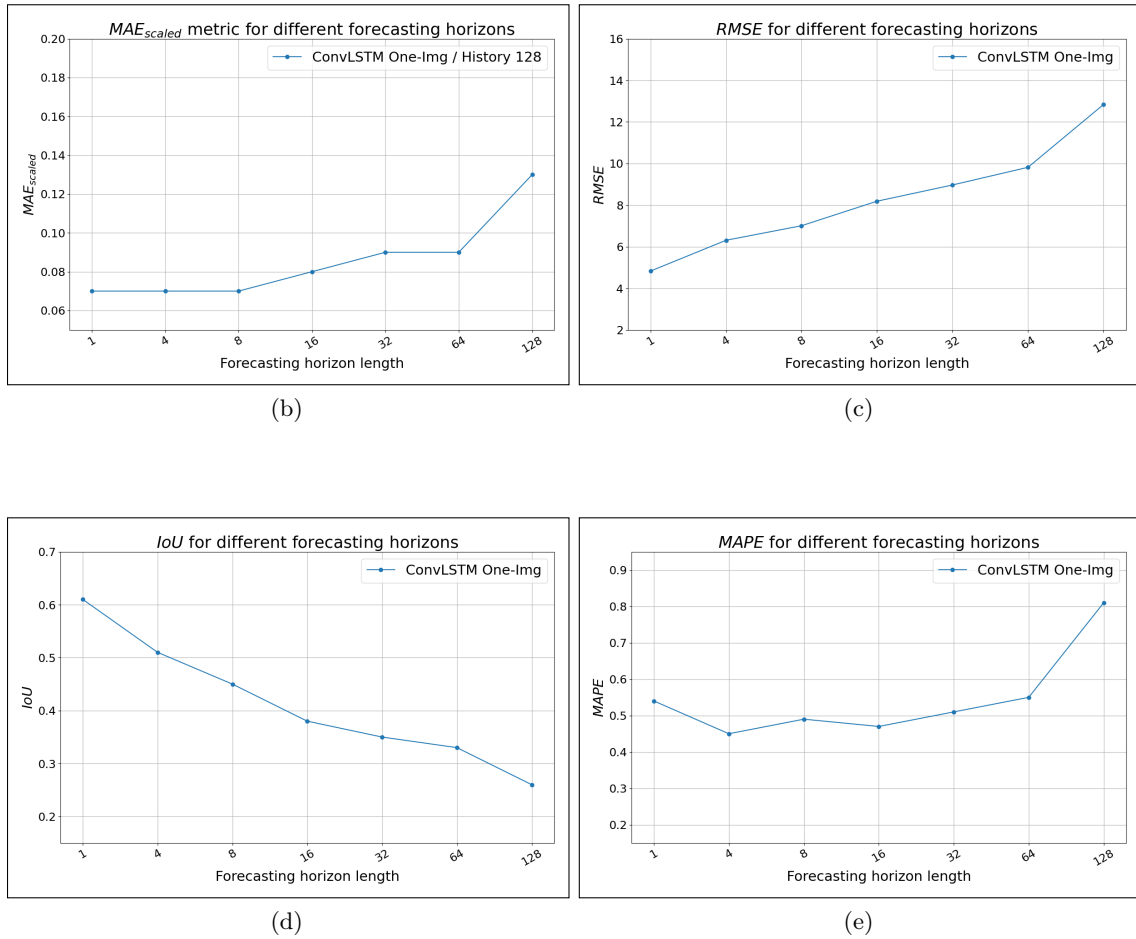


Figure 4.8: Plots using matplotlib for metrics results shown on Tables 4.5: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) RMSE (d) IoU (e) MAPE.

Section Takeaway:

In this section we explored how the *ConvLSTM-One-Img* model with a *forecasting history* of 128 and a *y-axis* size of 64 performs for **different prediction lengths**. We discussed how the problem is fundamentally different and much harder from predicting only the next value, which almost all related work focuses on. Similarly to the previous section, there is no determining factor in choosing an appropriate length, but more a balancing act of a good enough accuracy compared to the length of values predicted. After that, we explore our results for a range of **forecasting horizon** values from 1 to 128. As it was expected, we observed a decrease in Accuracy (and reduce in performance of the rest of the metrics) as we predict further into the future. In Chapter 5 we will mostly be comparing for **forecasting horizon** values in the range of 16 to 64.

4.8 Chapter Summary

In this chapter, we explored the use of computer vision machine learning models, based on the **ConvLSTM** module in order to learn memory access patterns and predict long-term into the future. At first we touched the subjects of how we would split the data in a full model deployment and how we performed it in our testing. After that, 2 models were proposed, the **ConvLSTM-Mult-Img** and the **ConvLSTM-One-Img**. Both models

have some parts of their architecture similar, with the core difference being that the latter model has a dense layer in between the down-sampling and up-sampling parts (similar to an Autoencoder). The **ConvLSTM-Mult-Img** model can take as an input multiple images and predict one image (of the same size in both axis). The **ConvLSTM-One-Img** model, due to the dense layer, accepts one input images of a variable length (*Forecasting History* length) and can produces an output images of a variable again length image (which can be different from the input) which is the *Forecasting Horizon*. The *y*-axis size in both cases remains the same between input and output. After that we presented how our performance metrics are calculated and why they where chosen. We note that the emphasis will be given on the **Accuracy** metric for the rest of our work.

We proceed by testing our models over different scenarios. At first the models performance was evaluated upon different **forecasting history** values (different number of images for the *ConvLSTM-Mult-Img* model and different length of input image for the *ConvLSTM-One-Img* model). It was clearly evident that the *ConvLSTM-One-Img* performs significantly better, with peak performance at a **forecasting history** value of **128** or **256**. Only this model was kept for further testing. After that an exploration was done for **different y-axis** size values. The *y-axis* size determines how much compression is performed when mapping the given page-range to a smaller set of classes, the pixels of the *y*-axis. Compromises need to be made between the compression that will be performed, thus the number of pages equivalent for each pixel, and *Accuracy*. The value was chosen based on a balancing between the later and eventually a **y-axis** size of **64** was chosen. Finally, after highlighting how the problem of predicting long-term into the future is fundamentally different and much harder from predicting only the next value, which almost all related work focuses on, we evaluated the performance for **different forecasting horizon** values. Similarly to the *y-axis* size exploration, we must accept the trade-off between *forecasting horizon length* and *Accuracy* and proceed with what suits us better. In our case we will continue with **forecasting horizon** values of **16** to **64** based on their high performance and sufficiently long prediction length.

Chapter 5

Comparison Against *LSTM* Forecasting Method

In Chapter 4, an alternative method for forecasting future pages in order to improve their prefetching was introduced. This solution leverages the power of computer vision and deep learning to learn patterns in previous data and forecast future possible missing pages. In this chapter, we are going to compare the model of the image-driven approach, the *ConvLSTM-One-Img* with the most common deep learning approach in the domain of time-series analysis and forecasting, the *LSTM*. There has been a recent explosion of interest for the use of *Transformer* based models in time-series forecasting as can be seen in [24, 76, 87, 50]. During the time this research was conducted, traditional *Transformers* suffered some fundamental limitations, e.g., they were generally not decomposable or interpretable, and most importantly for our work not very effective nor efficient for long-term forecasting. Nonetheless, as research progresses and advancements are made in *Transformer* architectures it is something to explore further and is included in the Future Work Chapter 6.3.

5.1 Traditional Time-series Forecasting Methods

As presented in Chapter 2, two of the most used approaches to time series forecasting is *ARIMA* and *Exponential Smoothing*. At first we experimented with *Exponential Smoothing*. We experimented by giving histories of 64 to 256 data-points and forecasting not that far in the future. It was immediately clear, even after some light tuning of the smoothing variables, that these models could perform averagely on short forecasting horizons, but as the requested forecasting horizon increased their performance dropped dramatically. Most of the times it seemed to fixate on one value and prediction only that one for all the future predicted values. In any case the results fall short by 10 – 20% in *Accuracy* compared to the *ConvLSTM-One-Img* model.

While Table 5.1 shows very low *Accuracy* when forecasting horizon is larger than 32, the rest of the metrics may have bad results but with an interesting observation. Metrics like *MAE_{scaled}*, *RMSE*, and *MAPE*, despite indicating a general poor performance, they all have similar if not better values to those, for example, in Table 4.5 for a forecasting horizon of 128, which had significantly better *Accuracy* and *IoU*. This discrepancy can likely be attributed to the nature of the *Exponential Smoothing* model. It tends to predict values close to the average, resulting in forecasts with fewer outliers. While this keeps individual errors (difference between prediction and actual value) from being very large, it also leads to consistently inaccurate predictions, reflected in the low *Accuracy* and *IoU*. Unlike the

other metrics (MAE_{scaled} , $RMSE$, and $MAPE$) that focus on individual prediction errors (magnitude of the difference), IoU captures the overlap between the predicted and actual values. This fundamental difference explains why IoU follows the low $Accuracy$ trend in Table 5.1, while the other metrics show seemingly comparable values to the higher-accuracy scenario.

We will not be doing an analysis on the size of the y-axis, or correspondingly to the $LSTM$ model the number of classes or discrete values each data point can have. All further testing will be done for a total of 64 classes.

<i>Exponential Smoothing</i>						
Forecasting Horizon	Accuracy	MAE_{scaled}	RMSE	IoU	MAPE	Inference Time pred/total
1	0.37	0.07	4.94	0.37	0.43	1.4ms/1781s
4	0.34	0.08	6.00	0.31	0.48	1.4ms/640s
8	0.30	0.08	6.84	0.27	1.53	1.5ms/226s
16	0.26	0.10	7.89	0.22	0.63	1.7ms/113s
32	0.22	0.11	08.98	0.19	0.73	1.5ms/42s
64	0.16	0.13	10.80	0.14	0.90	1.4ms/22s
128	0.10	0.16	13.21	0.07	1.04	1.6ms/14s

Table 5.1: Performance metrics of *Exponential Smoothing* for different values of forecasting horizon

<i>ARIMA</i>						
Forecasting Horizon	Accuracy	MAE_{scaled}	RMSE	IoU	MAPE	Inference Time pred/total
1	0.39	0.06	04.12	0.39	0.43	48ms/-
4	0.37	0.07	05.05	0.35	0.45	48ms/-
8	0.36	0.07	05.56	0.34	0.47	48ms/-
16	0.33	0.08	06.47	0.32	0.57	48ms/-
32	0.17	0.11	09.04	0.13	0.75	48ms/6200s
64	0.16	0.13	10.54	0.13	0.89	45ms/503s
128	0.09	0.16	12.95	0.07	1.03	45ms/130s

Table 5.2: Performance metrics of *ARIMA* for different values of forecasting horizon. We have "-" on the total time for inference of the model for the forecasting horizons of 16 and less because a smaller testing dataset (equally distributed through the dataset) was given due to the great time needed for inference

Similarly thought, as can be seen again on Table 5.2, the *ARIMA* model provided results following the same pattern as with *Exponential Smoothing*. An interesting thing to note is that while both *Exponential Smoothing* and *ARIMA* have small inference times their total inference time is a lot slower. This is due to the fact that there is no batch prediction. Furthermore the *ARIMA* model may train fast, but it is necessary to update the model in each iteration based on the past data, during the inference stage. This is not only very computationally expensive but also slow, thus it is not feasible in such a time sensitive scenario such as prefetching.

While both *ARIMA* and *Exponential Smoothing* are established timeseries forecasting techniques, they fall outside the scope of this thesis due to their focus on statistical methods. Given the unique nature of our problem and the established approach of employing a Computer Vision based approach, this thesis prioritizes comparing our model with other Machine Learning and Deep Learning models for a more relevant and comprehensive evaluation.

Example result of both *ARIMA* and *Exponential Smoothing* can be found in Figures 5.1.

These plots validate our observation that the models predict an average value, which leads to very bad *Accuracy* and *IoU* but not equally "bad" for the metrics of error.



(a) *Exponential Smoothing*



(b) *ARIMA*



(c) *GroundTruth*

Figure 5.1: Example plots of *Exponential Smoothing* and *ARIMA* predictions with the ground truth for a $fh = 32$. A total of 15 predictions are concatenated together for the each of the images

Section Takeaway:

Both **ARIMA** and **Exponential Smoothing** models results for all tested *forecasting horizons* fall short compared to our image-based approach with **10-20% less Accuracy** and even worse performance as the *forecasting horizon* grows. Interestingly, the performance metrics based on errors (MAE_{scaled} , $RMSE$, and $MAPE$) are not proportionately as bad as the *Accuracy* or *IoU* metrics are. This was explained due to the fact that both models, as proven by the prediction images included, predict an average value of their given history. Thus, especially in short forecasting horizon predictions, they may have a wrong prediction (low *Accuracy* and *IoU*) but the difference between prediction and ground truth is usually rather small, leading to the error based metrics to not perform as bad as expected. Further testing and experimentation with these models was not continued as neither did they perform competitively enough neither it is within our interest to explore them in more depth.

5.2 Current State-Of-The-Art Methods Proposed

In this section we are introducing the *LSTM* model we will be comparing our *ConvLSTM-One-Img* model with. *Autoencoder* or *LRCN* (*CNN* + *Autoencoder*) models had been explored during the searching done while selecting our proposed model (Section 4.3) and

did not provide competitive performance metrics in order to explore them further. Thus, this chapter is focused purely on the comparison with an *LSTM* model.

The structure of a *LSTM* model was described in Chapter 2. We implemented a rather simple *LSTM* in *Keras/Tensorflow*. The model has as input a one-dimensional timeseries input (of length equal to the Forecasting History) which goes through three *LSTM* layers and thereafter through two dense layers with the last one providing the output of a length corresponding to the Forecasting Horizon. The model can be seen in Figure 5.2.

In Chapter 3.2.2 we described the process of transforming the timeseries sequence $\{y_0, y_1, \dots, y_t\}$ to an image I_t ($\{y_0, y_1, \dots, y_t\} \rightarrow I_t$). At the last stage, before creating the 2D binary image, we have a sequence $\{\bar{y}_1, \bar{y}_2, \dots, \bar{y}_t, \bar{y}_{t+1}, \dots\}$ (as seen in Figure 3.9), this is the sequence we are providing to the *LSTM* input. Similarly, the labels provided for training for the *LSTM* is the corresponding sequence shifted *fh* in to the future.

Some experimentation and testing was conducted in order to tune the hyperparameters, such as the number of *layers* or the number of *units* in the *LSTM* modules. The *LSTM* solution was not researched and tested as thoroughly as possible, given that the main goal of our work is to give a general comparison of our proposed method and not compete with *State-Of-The-Art LSTM* applications. For example, the *Voyager* model [75] one of the highest performing ML prefetchers, out-competes our solution on multiple different aspects, but has a lot more complex and sophisticated handling of the problem, our solution is proposed as a proof of concept for further future research purposes. The optimal hyperparameters found for the *LSTM* model where found to be:

- Optimizer : Adam
- loss function : Categorical Crossentropy
- # of layers : 3
- # of units : 40
- Callbacks : *Early stopping* and *Reduce Learning Rate On Plateau*

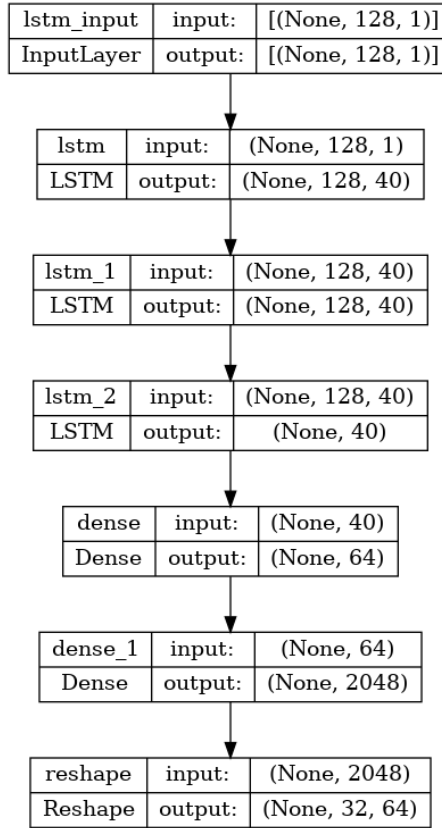


Figure 5.2: *LSTM* Model Architecture. The values and dimensions are affected by the input and output size of the model. In this case the *forecasting horizon* is 32 and *forecasting History* is 128.

We then proceed to perform the same experiments performed on Chapters 4.5 and 4.7 in order to compare in the next sections the different behavior of the models. As stated earlier we will be working with a value of 64 for the number of "classes" (equal to the y-axis size for the *ConvLSTM-One-Img* case).

First on Table 5.3 we have the performance metric results for testing for different *forecasting history* lengths, starting from 16 and reaching 768 datapoints.

Input	Accuracy	MAE _{scaled}	RMSE	IoU	MAPE	inference Time [ms]
16	0.21	0.12	11.12	0.15	0.88	42.6/1.83
32	0.37	0.10	10.33	0.27	0.72	43.9/2.09
64	0.40	0.10	09.97	0.29	0.66	45.86/3.34
128	0.43	0.09	09.96	0.29	0.63	48.53/4.51
256	0.41	0.09	09.89	0.29	0.60	54.16/7.78
512	0.39	0.10	10.15	0.29	0.61	67.70/14.96
768	0.37	0.11	10.61	0.27	0.63	64.49/22.12

Table 5.3: Performance metrics as forecasting history changes for the *LSTM* model for $fh = 64$.

On Table 5.4 we have the results after testing for the *forecasting horizon* lengths of 1, 2, 4, 8, 16, 32, 64 and 128.

Forecasting Horizon	Accuracy	MAE _{scaled}	RMSE	IoU	MAPE	inference Time [ms]
1	0.67	0.06	03.77	0.67	0.39	45.39/4.31
4	0.58	0.07	05.86	0.50	0.45	47.76/4.40
8	0.56	0.07	06.70	0.47	0.49	45.45/4.36
16	0.53	0.08	08.12	0.44	0.55	46.64/4.29
32	0.47	0.08	08.43	0.35	0.50	48.86/4.44
64	0.43	0.10	09.97	0.29	0.60	47.08/4.35
128	0.37	0.10	10.94	0.26	0.68	49.14/4.45

Table 5.4: Performance metrics as forecasting horizon changes for the *LSTM* model with a forecasting history of 128 as input.

5.3 Comparison on History

By observing the results on Table 5.3 they seem to follow a similar trend as seen with Table 4.2. Both models display better performance as the forecasting history increases and peaking when the history is $2 \times - 4 \times$ the forecasting horizon value, so both models at around 128 and 256. The plots in the Figure 5.3 display the performance metrics results for the *LSTM* and *ConvLSTM-One-Img* models for the different *forecasting history* values. In general by observing the plots in Figure 5.3 we can see that both models follow a similar trend in most performance metrics with the most noticeable difference being that the *LSTM* model requires a smaller forecasting history in order to start performing well. Especially observing the *Accuracy* and *IoU* metrics it clear how the *LSTM* model increases immediately when the *forecasting horizon* reaches a value of 32 surpassing the *Accuracy* of the *ConvLSTM* model. In comparison the *ConvLSTM-One-Img*, catches up in performance when the history has reached a value close to 128 where it exceeds the performance of the *LSTM* model from that point and after. The *MAPE* plot further reveals *ConvLSTM-One-Img* model has a lower value for all different histories. Its consistently lower *MAPE* values, compared to the *LSTM* model, suggest the latter generates more outliers. This is likely because, while both models have similar *RMSE*, *MAPE* penalizes larger errors more severely.

Additionally we must comment on the *Inference Time* of the *LSTM* model. As already observed in Chapter 4.5 the *ConvLSTM* model Inference time, does not increase when we provide an longer input. On the other hand, as can clearly be seen on the (*f*) plot in Figure 5.3, the *LSTM* model starts with an *Inference Time* less than half of that of *ConvLSTM-One-Img*. But, in contrast with the later, as the input increases so does the inference time significantly. However, given that the *LSTM* model provides the best *Accuracy* for a *forecasting history* of 128, for that value and smaller histories the *LSTM* has a lower inference time compared to the *ConvLSTM* based model.

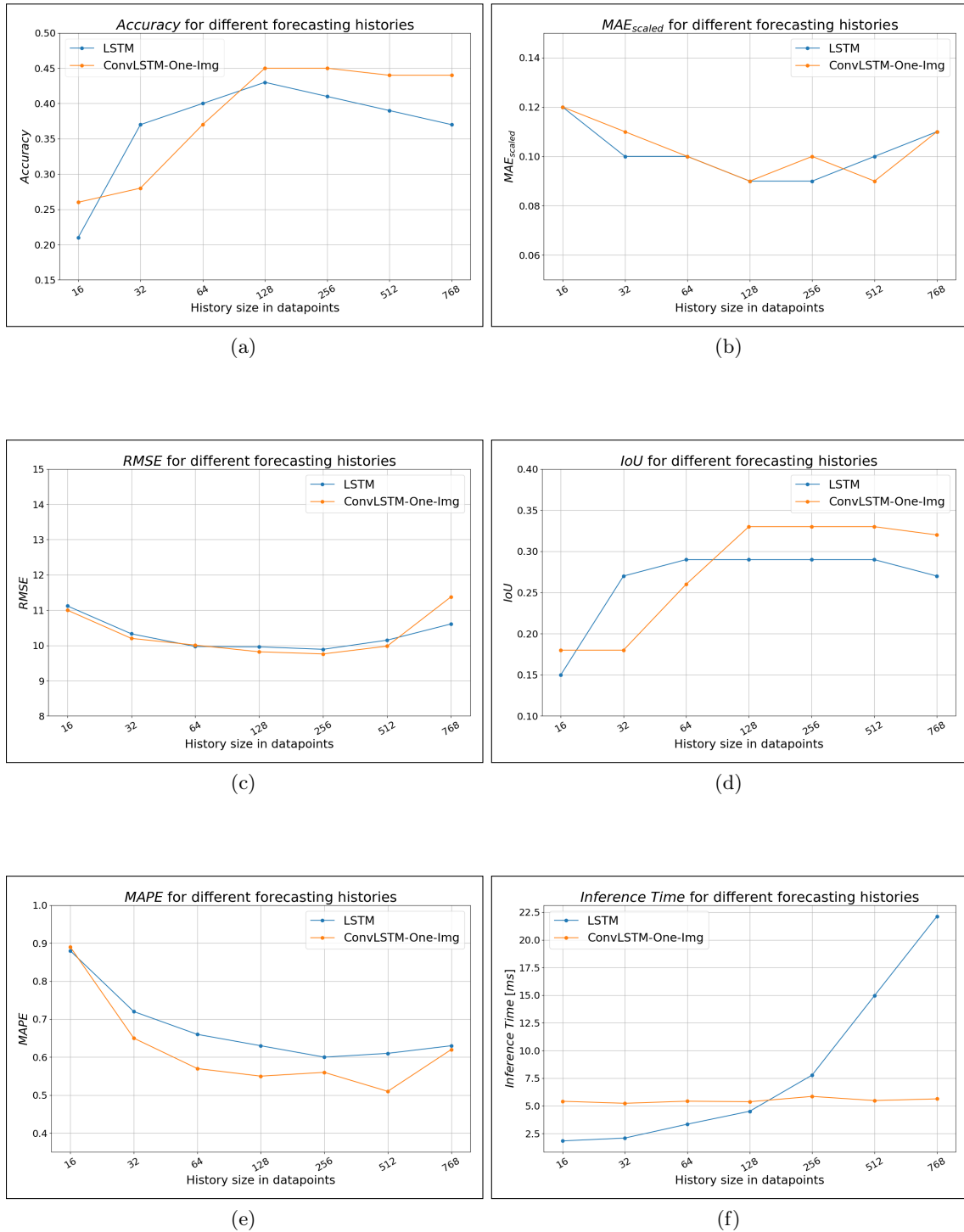


Figure 5.3: Plots using matplotlib for performance metrics results shown on Table 5.3 for LSTM model: (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) RMSE (d) IoU (e) MAPE (f) Inference Time.

5.4 Comparison on Forecasting Horizon

On Chapter 4.7 we analyzed the importance of selecting a proper forecasting horizon and we highlighted the importance in understanding that forecasting 1 timestamp ahead is almost an entirely different problem from forecasting multiple timestamps in to the future.

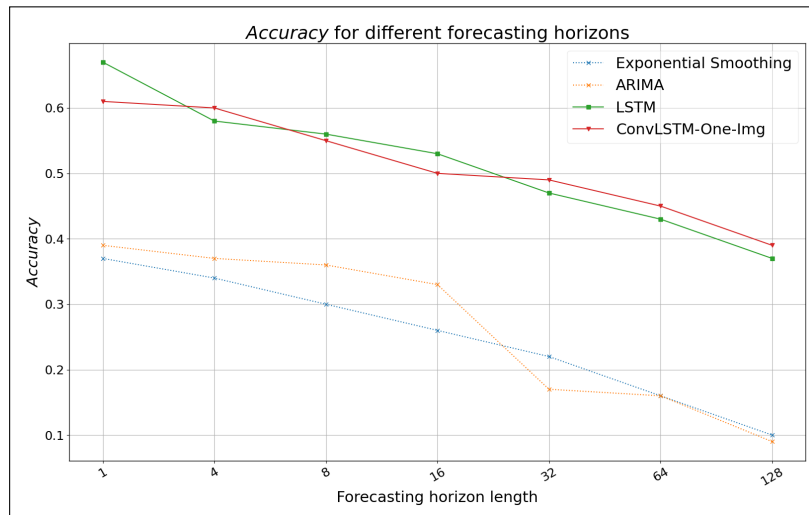
By examining Table 5.4, as we expected, the *LSTM* model seems to perform very well on the short forecasting horizons as this is known from past work to be one of the models' great strengths in timeseries prediction. It must be noted, that most likely, even better performance could be achieved for the very short-term prediction length. But given our goal was to explore the problem of long-term forecasting, the work conducted was in order to develop and optimize the model accordingly. With the increase of the distance of the prediction there is some loss in performance, which is acceptable given how much more complex the problem has become.

The plots on Figure 5.4 provide a visual representation for each of our performance metrics for the *LSTM*, *ConvLSTM-One-Img*, *Exponential Smoothing* and *ARIMA* models. As can be clearly seen both Machine Learning models outperform by a significant margin the results of *Exponential Smoothing* and *ARIMA*. We can confirm our previous observation, that for small forecasting horizon values (< 16) even though *Exponential Smoothing* and *ARIMA* have significantly less *Accuracy* and *IoU* values, the error metrics (MAE_{scaled} , *RMSE* and *MAPE*) are very similar to our neural network models. Most likely due to the fact that these models seem to predict just an average value of its given history and thus not leading to great error when the forecasting horizon is small. Of course as we predict further in to the future, the averaged value leads to an increase in outliers and that would explain the significant increase of the *MAPE* error. As for the models we developed, they seem to follow almost an identical trend with minor discrepancies in between different metrics and forecasting horizons. The latter could possibly imply that this is due to a limitation of our input dataset provided. More specifically, as explained in Chapter 3.2.2, after applying our mapping and binning of the given timeseries data we have created a sequence $\{\dots, \bar{y}_{t-1}, \bar{y}_t, \bar{y}_{t+1}, \dots\}$. For the case of the *LSTM* model this is its input, for the *ConvLSTM-One-Img* model we do a 2D representation to a binary image (explained analytically in that Chapter). There are high chances that our current information mapping approach might not be ideal for our given problems with frequent, seemingly random access patterns. This could be a limit the learning potential of both the tested models and the reason they perform so similarly. More information for other directions that can be followed are found in the Future Work Chapter 6.3. Similar to the *ConvLSTM-One-Img* model, for different forecasting horizons (different output dimension) there is no increase in the inference time.

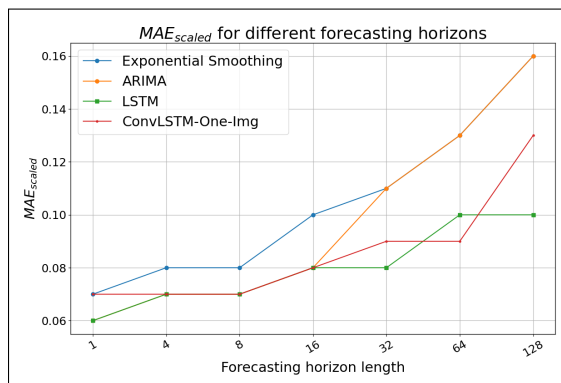
In Figure 5.5 and 5.6 we have two examples of the prediction of the two machine learning models compared to the *Ground Truth* value. On the first example, the total predictions are "harder" given the different strides occurring and we see how both of our models perform modestly. On the other hand, interestingly, examining the second example, the ground truth seems almost like two parallel lines (we know that there are never two values per column). The *ConvLSTM-One-Img* model seems to predict only the upper line and even so achieves an acceptable *Accuracy* of 0.48. However, while the *LSTM* seems to perform similarly, visually, by inspecting the image values (opening the image as an array in *Python* etc.) we notice that the predicted results are one or two pixels far from the true prediction, thus explaining the very low *Accuracy* of 0.16 on a seemingly "simple" prediction.

Figures 5.5 and 5.6 illustrate the predictions of the two machine learning models compared to the *Ground Truth* value. The first example presents a more challenging scenario with significant variations (strides) in the data. Here, both models achieve modest performance. In Figure 5.6, the *Ground Truth* appears to consist of two parallel lines (although we know there is only one value per column). The *ConvLSTM-One-Img* model captures only the upper line but still achieves an acceptable *Accuracy* of 0.48. The *LSTM* model, seems to perform similarly on a visual inspection. However, by examining the image values directly (e.g., opening the image as an array in *Python*), we discover that the predicted results are one

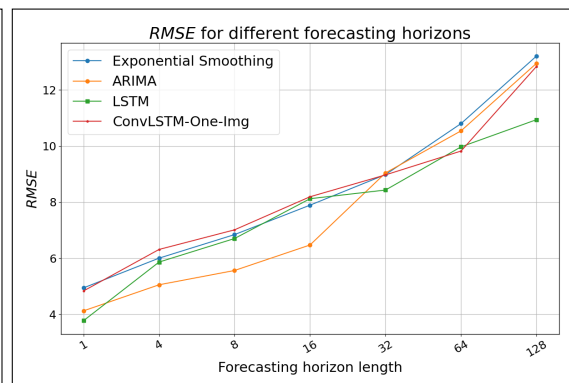
or two pixels off from the true values. This explains the significantly lower accuracy of 0.16 for the *LSTM* model, despite the seemingly 'simple' prediction task. Thus we can see the inspecting visually the results can provide us with a small intuition about the performance of the model. But given how easily a "good" predictions turns out to be inaccurate we should rely on the metrics. Furthermore, given the huge size of the dataset and different behavior of the patterns within it, it is hard to generalize about the results by viewing a couple of images.



(g)



(h)



(i)

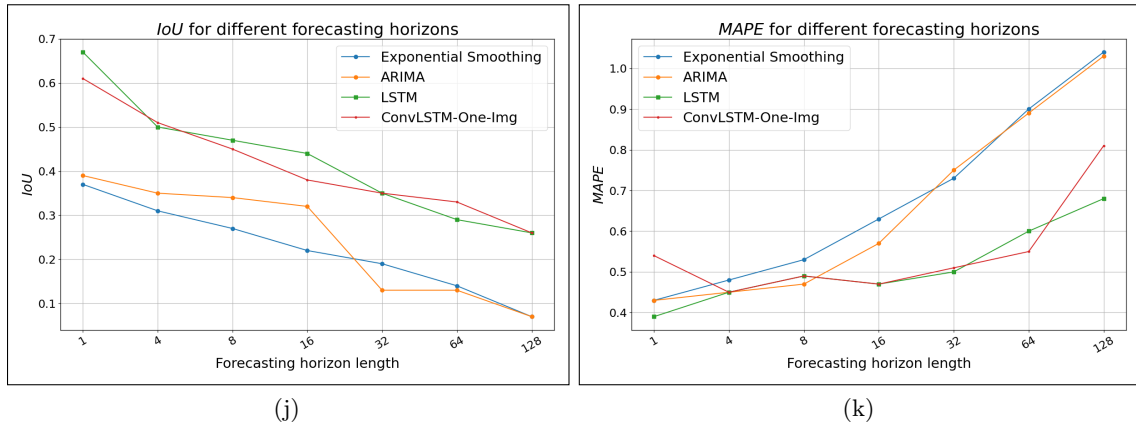


Figure 5.4: Plots using `matplotlib` for metrics results shown on Tables 4.5, 4.7, 5.1 and 5.2 comparing the *Exponential Smoothing*, *ARIMA*, *LSTM* and *ConvLSTM-One-Img* models: (a) *Accuracy/Dice Coefficient* (b) *MAE_{scaled}* (c) *RMSE* (d) *IoU* (e) *MAPE*.



(a) *ConvLSTM-One-Img*



(b) *LSTM*



(c) *Ground Truth*

Figure 5.5: Example plots of *ConvLSTM-One-Img* (*Accuracy*=0.36) and *LSTM* (*Accuracy*=0.30) predictions compared with the *Ground Truth* for a $fh = 32$ and a history of 128 data points. A total of 16 predictions are concatenated together for the each of the images creating an image of $64 \times 512 \times 1$ image.



(a) *ConvLSTM-One-Img*



(b) *LSTM*



(c) *Ground Truth*

Figure 5.6: Example plots of *ConvLSTM-One-Img* (*Accuracy*=0.48) and *LSTM* (*Accuracy*=0.16) predictions compared with the *Ground Truth* for a $fh = 32$ and a history of 128 data points. A total of 16 predictions are concatenated together for the each of the images creating an image of $64 \times 512 \times 1$ image.

Section Takeaway:

In this section we explore how the **LSTM** and the **ConvLSTM-One-Img** models **compare for different forecasting horizon** values. We include the statistical methods mentioned earlier, *Exponential Smoothing* and *ARIMA*. The later models perform significantly worse than both of the machine learning ones implemented and are discarded from further reference. The *LSTM* and *ConvLSTM-One-Img* models have very comparable results and follow an almost identical pattern throughout all the tested values. A possible explanation for the extremely similar results could be the visualization chosen, and thus the limiting factor for both models being the dataset and not their capabilities to interpret and predict data.

5.5 Comparison on Training and Inference Times

To leverage prefetching, models must be compact, efficient, and prioritize rapid inference for real-time performance in order to avoid stale prefetches. As discussed in Chapter 1, we aren't aiming to outperform hardware prefetchers in any single of the above metrics. Instead, this thesis explores the potential of computer vision for prefetching, focusing on the concept rather than a practical implementation at this stage. The numbers presented in Table 5.5 are provided just for reference. It is well known that we exceed by orders of magnitude the very tight specifications needed. Furthermore, we did not explore techniques to reduce the model sizes memory footprint, by implementing techniques such as pruning [5, 40] or quantization of weights to a lower number of bits (e.g., from 32 to 8 bits) [92]. As mentioned in 1.2, MemMAP [78] proposes extremely compact *LSTM* models that can

predict the next memory access with high accuracy among other added benefits of their approach. Machine Learning solutions are still far from achieving the size and especially the speed and computation constrains need in such applications. However, advancements has been achieved the recent years towards machine learning solution closer to hardware implementations by limiting computation costs [62] and minimizing their size [30, 70].

Comparison on Training Time, Size and # Parameters				
Model	fh	Training Time [m]	model size [MB]	# Parameters
<i>LSTM</i>	16	68	1.12	101k
	32	36	1.49	168k
	64	24	2.23	301k
<i>ConvLSTM</i>	16	32	49.43	4M
	32	20	67.00	6M
	64	12	101.99	10M

Table 5.5: Training time, model size and number of trainable parameters for *ConvLSTM-One-Img* and *LSTM* model. Both models forecasting history is 128.

Interestingly, the Table (5.5) shows that the *LSTM* model, despite having significantly fewer parameters to train, requires a little bit more time for training. While using *TensorBoard*, *TensorFlow's* visualization toolkit, we noticed the *ConvLSTM-One-Img* model rapidly learned (in a matter of a few epochs) the training data, but led to overfitting (Figure 5.7). That is when added measures such as *Early Stopping* kicked in and stopped further training. In contrast, the *LSTM* model converged on a solution gradually but slowly. While each training epoch took less time, the model required many more epochs to achieve good results, eventually leading to longer overall training times. Again, *Early Stopping* typically activated before the maximum number of epochs was reached.

About the inference time, as discussed earlier in Figure 5.3, plot (f), we can observe the different behavior between the two models as forecasting history increases. More specifically, the *ConvLSTM-One-Img* model exhibits no change in its inference time regardless of whether the image is 16 timestamps wide or 768. On the other hand, the *LSTM* model is affected by the input size. As the input size increases, so does the model's inference time. For the forecasting history of 128, where both models perform best, they have similar inference times, with the *LSTM* model being slightly faster (approximately 0.7 milliseconds less). This shows, how the *CNN* based model efficient architecture for parallel processing on grid-like data makes it behave the same no matter the input length. Notably, both models remain unaffected for all tested forecasting horizons.

It's important to consider that, as we have a very powerful GPU (*NVIDIA A100-PCIE-40GB*) with *CUDA* enabled, training times and inference speeds are most likely significantly faster compared to using a commercially available GPU for model training.

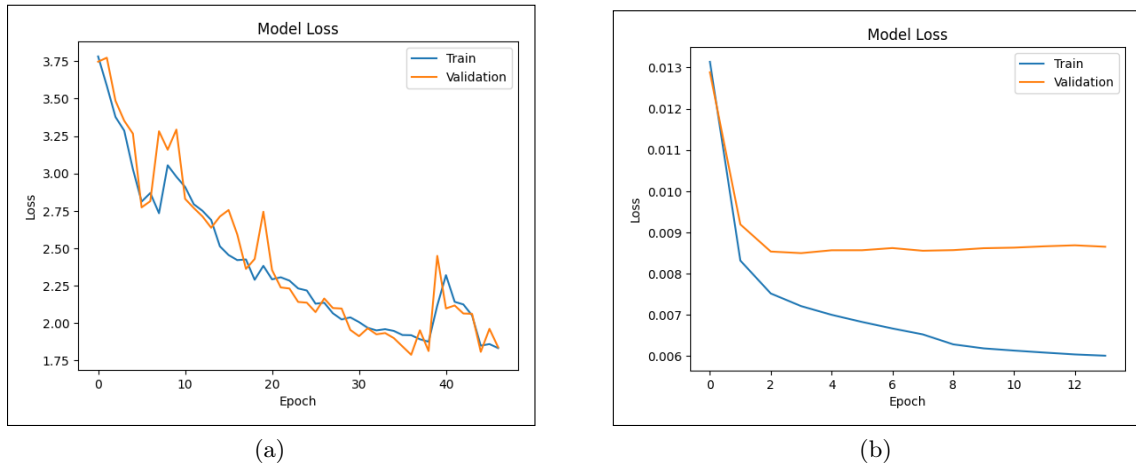


Figure 5.7: Loss function plots for (a) *LSTM* and (b) *ConvLSTM-One-Img* model.

Section Takeaway:

In this section we compare the *LSTM* and the *ConvLSTM-One-Img* on their size, complexity (number of parameters), training and inference time. Both models are not optimized for either storage or speed. On the training time, interestingly even though the *LSTM* has orders of magnitude less trainable parameters its training is a lot slower. On the other hand the *ConvLSTM* based model, trains extremely fast, with the drawback of starting to overfit quite early on the training. On the *Inference time* we discussed how the two models behave differently when experimenting on forecasting histories, with the *ConvLSTM* model seeing no change in its *Inference Time*, and the *LSTM* model seeing an increase the longer the forecasting history is. However, within the values we are interested for the *LSTM* model (128 and less) it performs slightly better than the image-based model.

5.6 Comparison Throughout the Dataset

Testing Results Throughout the Dataset

As explained in the Chapter 4.2 all of the above models have been trained on 10 – 30% of the *bc* benchmark, and then tested on the rest of it. Of course this leads to worse results because as seen in Figure 4.1 the benchmark changes behavior throughout its execution, leading to the use of a model in data patterns that are unseen. In order to see how our models perform as we go through the benchmark we tested independently every 10%. We tested with both *LSTM* and *ConvLSTM-One-Img* models with an input history of 128 and a forecasting horizon of 32 or 64. Results for all performance metrics can be found in Tables 5.6, 5.7 and 5.8 and their corresponding visualization in Figure 5.8. Observing the results two important things stand out. First of all it is evident from all plots, that at first that they are some changes in the dataset from 30 – 50% which has some negative impact on the performance metrics. After which, for the next 20% both models perform almost identically until the we go to the last quarter of the dataset. There after both models struggle to handle any changes that are occurring and start to perform a lot worse, reaching very large error values and very low *Accuracy*. It must be noted that in all metrics aside from the *Accuracy* the *LSTM* model seems to perform slightly better. But, due to the nature of our problem, the most important metric to evaluate upon, is the *Accuracy* as

all other metric give us an estimation of how close our results are. In practice, from all of our metrics, only the *Accuracy* metric indicates a higher performing prefetcher. Based on the above and the slight edge of the *ConvLSTM-One-Img* model on "new" parts of the dataset makes our model relevant.

Accuracy								
<i>fh</i>	<i>Model</i>	30%	40%	50%	60%	70%	80%	90%
32	<i>LSTM</i>	0.57	0.51	0.43	0.40	0.40	0.27	0.19
	<i>ConvLSTM</i>	0.56	0.49	0.42	0.41	0.42	0.30	0.22
64	<i>LSTM</i>	0.53	0.46	0.41	0.38	0.38	0.25	0.18
	<i>ConvLSTM</i>	0.53	0.47	0.41	0.39	0.40	0.27	0.19

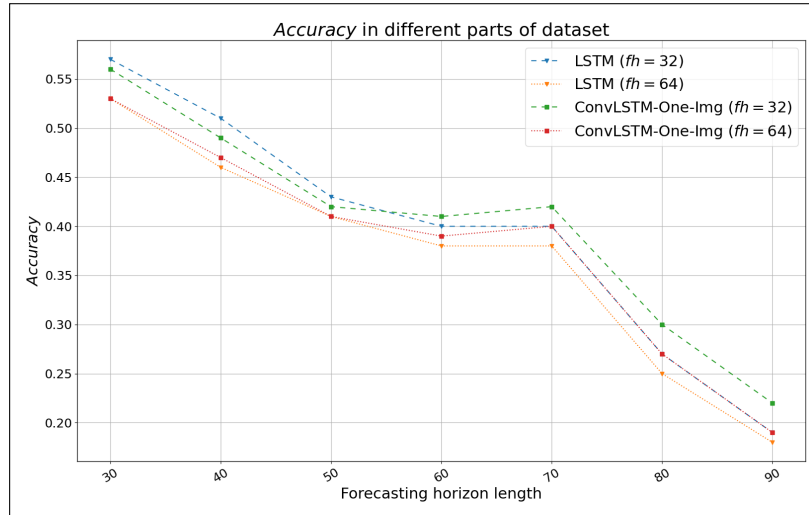
Table 5.6: *Accuracy* performance on different parts of the dataset

MAE _{scaled}									RMSE								
<i>fh</i>	<i>Model</i>	30%	40%	50%	60%	70%	80%	90%	<i>fh</i>	<i>Model</i>	30%	40%	50%	60%	70%	80%	90%
32	<i>LSTM</i>	0.07	0.07	0.08	0.09	0.09	0.11	0.13	32	<i>LSTM</i>	7.57	8.33	8.91	9.10	9.30	11.05	13.50
	<i>ConvLSTM</i>	0.07	0.08	0.09	0.10	0.10	0.13	0.17		<i>ConvLSTM</i>	7.85	8.92	9.86	10.08	10.23	12.37	15.41
64	<i>LSTM</i>	0.07	0.08	0.09	0.09	0.10	0.13	0.16	64	<i>LSTM</i>	8.46	9.43	10.24	10.33	11.05	12.10	15.20
	<i>ConvLSTM</i>	0.08	0.09	0.11	0.11	0.11	0.14	0.18		<i>ConvLSTM</i>	8.93	10.34	11.56	11.86	11.94	14.69	18.30

Table 5.7: *MAE_{scaled}* and *RMSE* metrics performance on different parts of the dataset

IoU									MAPE								
<i>fh</i>	<i>Model</i>	30%	40%	50%	60%	70%	80%	90%	<i>fh</i>	<i>Model</i>	30%	40%	50%	60%	70%	80%	90%
32	<i>LSTM</i>	0.47	0.40	0.35	0.34	0.34	0.24	0.16	32	<i>LSTM</i>	0.47	0.51	0.52	0.52	0.52	0.57	0.63
	<i>ConvLSTM</i>	0.44	0.37	0.32	0.30	0.30	0.20	0.14		<i>ConvLSTM</i>	0.48	0.51	0.54	0.54	0.54	0.60	0.68
64	<i>LSTM</i>	0.40	0.34	0.30	0.28	0.27	0.19	0.12	64	<i>LSTM</i>	0.54	0.59	0.60	0.61	0.62	0.66	0.75
	<i>ConvLSTM</i>	0.41	0.36	0.29	0.28	0.28	0.18	0.11		<i>ConvLSTM</i>	0.54	0.60	0.65	0.65	0.66	0.79	0.93

Table 5.8: *IoU* and *MAPE* metrics performance on different parts of the dataset



(a)

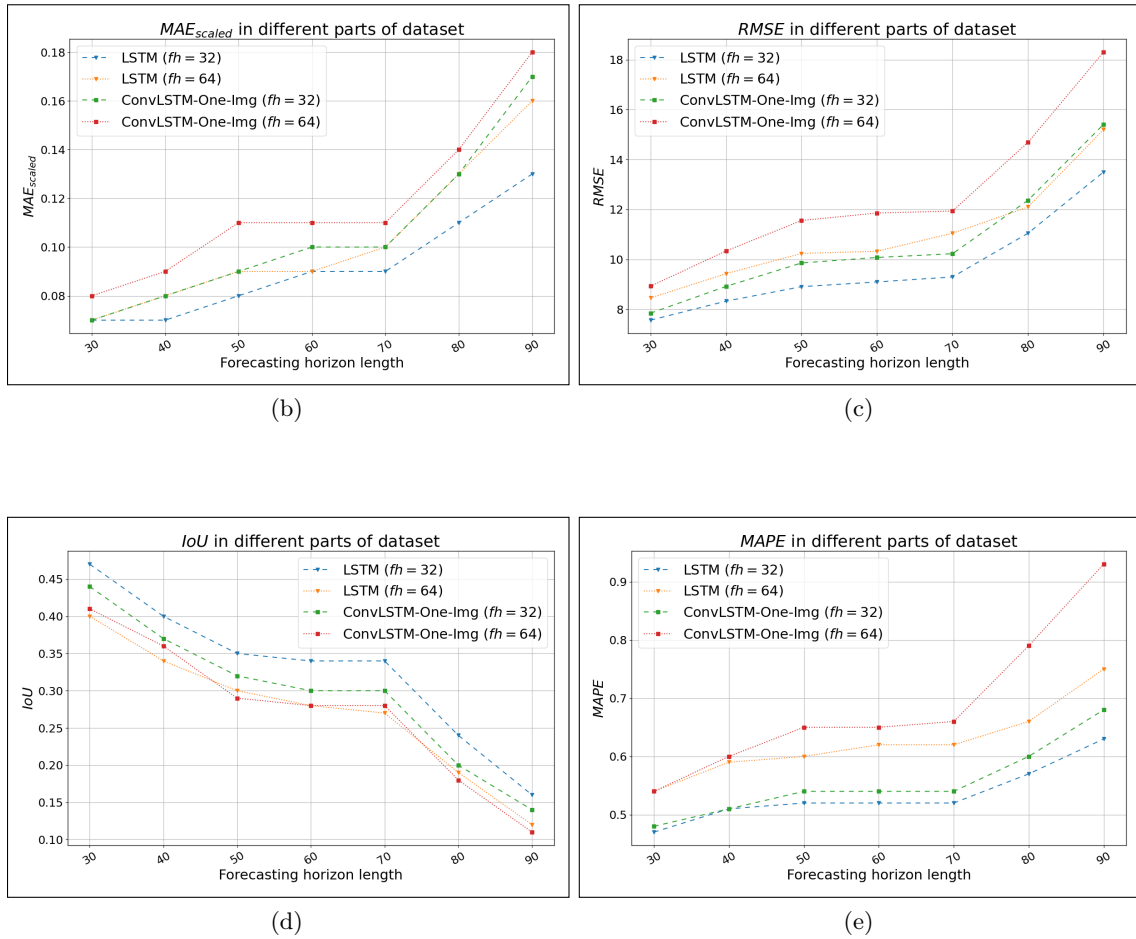


Figure 5.8: Plots using matplotlib for metrics results shown on Tables 5.6, 5.7 and 5.8 for *LSTM* and *ConvLSTM-One-Img* models with a forecasting history of 128 (a) Accuracy/Dice Coefficient (b) MAE_{scaled} (c) *RMSE* (d) *IoU* (e) *MAPE*.

Training and Testing on Different Parts of the Dataset

One of the ways in approaching the problem of prefetching with Machine Learning models is to train your model online while the app is executing [57, 77, 75]. To be more specific, in the cases where they have not trained the model offline, an approach of having periods of training and periods of predicting during execution time is performed. One approach to tackling the prefetching problem with machine learning models involves training the model online during app execution [57, 77, 75]. Specifically, for these type of scenarios (where offline training hasn't been performed), this method usually includes periods of training in order to learn data access patterns with periods of prediction. By mimicking this concept, we can achieve two goals: first to delve into the effectiveness of our model on the latter part of the dataset, where both of our models faced difficulties, and explore this concept itself. As mentioned in Chapter 4.2 this method comes close to the idea of blocked cross-validation, which we did not implement in order to explore the results separately. To test the above in Table[] we have have the results of training and testing the *LSTM* and *ConvLSTM-One-Img* model with a history of 128 and a forecasting horizon of 32 on these 3 cases:

- Training on: 10 – 30% - Testing on: 30 – 40%

- Training on: 30 – 50% - Testing on: 50 – 60%
- Training on: 60 – 80% - Testing on: 80 – 90%

From Table 5.9 we can see that our *ConvLSTM-One-Img* model slightly outperforms the *LSTM* model as we proceed further into the dataset.

<i>Model</i>	Train	Test	Accuracy	MAE_{scaled}	RMSE	IoU	MAPE
LSTM	10-30%	30-40%	0.55	0.07	7.58	0.47	0.45
	30-50%	50-60%	0.46	0.08	8.87	0.33	0.56
	60-80%	80-90%	0.35	0.12	10.38	0.25	0.61
ConvLSTM	10-30%	30-40%	0.54	0.07	8.19	0.43	0.49
	30-50%	50-60%	0.47	0.09	9.38	0.35	0.53
	60-80%	80-90%	0.37	0.11	10.14	0.29	0.56

Table 5.9: Performance metrics of *LSTM* and *ConvLSTM* models trained and tested on different parts of the dataset. Forecasting History is 128 and Forecasting Horizon 32.

Section Takeaway:

In this section, we tested the behavior of our two machine learning based models in 2 different scenarios. At first we explored testing a model trained on 10-30% of the dataset and tested on the rest of it, visualizing how the performance metrics deteriorate throughout the testing. We found that in the first parts of the testing set, immediately after the train set, the *LSTM* model performance slightly better. As we proceed further in to the dataset, where the information starts to have different patterns and behavior both models performance is worse. The *ConvLSTM-One-Img* model though, has a slightly better *Accuracy* at these later parts of the dataset, suggesting a stronger capability of generalizing compared to the *LSTM* model. After that, we train the models on 3 different parts of the dataset and test on part immediately after. Even though we trained again the data, we can see that the *ConvLSTM-One-Img* model, again surpasses by a small margin the *LSTM* model on the 2 later parts of the dataset.

5.7 Chapter Summary

In this chapter, we compared the model of the image-drive approach, the *ConvLSTM-One-Img* with the traditional, and the most common deep learning approach in the domain of time-series analysis and forecasting, the *LSTM*. At first the performance for our problem was evaluated for the *ARIMA* and *Exponential Smoothing* methods. These solutions proved inadequate for the complex data, demonstrating significantly lower accuracy (10-20% less) compared to the proposed image-based model. Consequently, further exploration of these solutions was deemed unnecessary. Subsequently, the chapter dives into the structure, hyperparameters, and selection process of the *LSTM* based model. This section also analyzes the effectiveness of the *LSTM* model under various configurations providing the performance metrics for different combinations of forecasting history length and horizon. In the next to subchapters, a comparison is done between the *LSTM* and *ConvLSTM* models. Plots depicting their performance as forecasting history or horizon changes are included. When comparing for different forecasting histories an observation is done that the *LSTM* model can learn from a smaller history, at around 32 timestamps, in order to start performing, while the *ConvLSTM-One-Img* model needed 128 data points to be on par. For longer histories our model surpasses the *LSTM* model in performance. After

that, a comparison on the forecasting horizon is presented, including the statistical models explored in the beginning of the chapter. It is clear how those models underperform and the *LSTM* and *ConvLSTM-One-Img* models have very comparable results and follow an almost identical pattern throughout all the tested values. A possible explanation for the extreme similarity in the results is given based on the visualization chosen, that could be a limiting factor for both models. Furthermore, we briefly elaborate on the size, number of parameters, training and inference time of the models. Interestingly even though the *LSTM* has orders of magnitude less trainable parameters its training is a lot slower. On the other hand the *ConvLSTM* based model, trains extremely fast, with the drawback of starting to overfit quite early on the training. Finally, we tested the behavior of our two machine learning based models in 2 different scenarios. At first we explored testing a model trained on 10-30% of the dataset and tested on the rest of it, and visualize how the performance metrics deteriorate throughout the testing. We found that in the first parts of the testing set, immediately after the train set, the *LSTM* model performance is slightly better. As we proceed further in to the dataset, where the information starts to have different patterns and behavior both models performance deteriorates. The *ConvLSTM-One-Img* model, has a slightly better *Accuracy* at these later parts of the dataset, suggesting a stronger capability of generalizing compared to the *LSTM* model. After that, we train the models on 3 different parts of the dataset and test on part immediately after. Even though we trained again the data, we can see that the *ConvLSTM-One-Img* model, again surpasses by a small margin the *LSTM* model on the 2 later parts of the dataset.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

This thesis presented a novel approach for learning memory access patterns and forecasting multiple future pages, by leveraging state-of-the-art computer vision and machine learning methods. Modern computing systems, have witnessed a significant rise in processing power. This remarkable progress, has created a growing disparity between compute power and memory access speed, leading to performance bottlenecks. Current approaches, including hardware-based data prefetching and system-level Hybrid Memory Systems, have limitations that obstruct their ability to fully address the growing demands for efficient data management. By introduction information by an accurate and robust page predictor the mentioned implementations could increase their performance significantly and hide the speed difference between CPU and memory.

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 and computer vision machine learning approaches to learn and eventually forecast future pages needed for an application execution.

The initial analysis started by visualizing memory access trace files, provided by a prefetching competition. Focus was given on a specific area of specific benchmarks, where most of the information was within a specific section. We propose an image-based visualization pipeline, that utilizes a defined set of rules to visually represent the data and outlines methods for decomposing these images to the corresponding prediction. The above pipeline is used in tandem with image-based machine learning models, where the visualized data is provided as an input to the model in order to predict an image, the output image is then decomposed to its numerical data that correspond to the predicted future page accesses. A couple of image-based machine learning solutions were explored and after some experimentation of the various parameters that affect the prediction, such as the compression applied and forecasting history and forecasting horizon, only the higher performer model was kept.

We then compared with traditional statistical timeseries forecasting methods and after that with the most common state-of-the-art machine learning timeseries forecasting approach, the *LSTM* model. Our analysis shows that our image based predictor can forecast future page access up to some accuracy, far surpassing the statistical based models and being very comparable with the sequence-to-sequence based *LSTM* model. In conclusion, this thesis identifies and demonstrates how the use of an image-based pipeline for predicting future page access, can perform nearly identically with current state-of-the-art proposed methods.

In some scenarios, where a longer history is given or the dataset changes its patterns to some extent, our model outperforms the *LSTM* approach. This thesis aims to lay the foundations for supporting future use of computer vision, visualization and image-based pipelines inside systems software with a great variety of future approaches. Coincidentally, very recent work that was proposed after our research, validates our approach by proposing a state-of-the-art image based solution that outperforms previous non-image machine learning solutions on the problem of data prefetching one point into the future [57].

6.2 Challenges and Limitations

During our work there were a few challenges and limitations that we had to deal and compromise with. They are presented briefly in the next paragraphs.

Input Augmentation and Visualization

Earlier on this thesis we emphasized on the importance of effectively transforming raw data into images for accurate predictions within image-based machine learning pipelines. Our approach primarily focuses on visualizing the time series data "in its natural state" as two-dimensional images. In these images, the x-axis represents individual timestamps, and the y-axis reflects the corresponding page range (with some "compression" applied). Unfortunately this visualization does not include other useful information, that could be valuable for our models. As mentioned in Chapter 3 a great amount of effort was put in order to find the appropriate visualization. During the first months of our work, we had emphasized in a visualization that did not produce 1 data point per column on the image, but focused on depicting a whole window of a specific *cycle counts* length within the given visualization. A whole set of different *ConvLSTM*, *Autoencoder* and *LSTM* models were developed and experimented with. This approach was eventually dropped given that we already have some compression on the y-axis, by including further on the x-axis would make it very hard to compare our results in terms of performance and especially accuracy given the multiple prediction per column (which would also introduce the need of researching a proper threshold value etc.). Eventually the later we dropped this visualization and proceeded with the one proposed in our thesis.

As discussed in Chapter 5.4 both models developed and compared, seem to follow almost an identical trend with minor discrepancies in between different metrics and forecasting horizons. The later could possibly imply that this is due to a limitation of our input dataset provided. More specifically, as explained in Chapter 3.2.2, both models take the same input, with the only difference being the 2D visualization for the *ConvLSTM* model. There is a high possibility that our current information mapping approach might not be ideal for our given problems with frequent, seemingly random access patterns. This could be a limit for the learning potential of both the tested models and the reason they perform so similarly as they both "learn" as much as the dataset allows them. Information for other directions that can be followed in order to enhance the information of the input dataset, are found in the Future Work Chapter 6.3.

Overfitting

Our image based *ConvLSTM-One-Img* model suffered greatly from overfitting. Methods such as *dropout* layers, *Early stopping*, *Reduce Learning Rate on Plateau* etc. were implemented in order to prevent the model from overfitting. All these techniques had some

valuable impact, but the model still tended to overlearn the training data and thus possibly limiting its abilities for generalizing better for future testing data. It is very promising how our model learns extremely fast the given training data, specifically in matter of a couple of epochs most of the training is performed, highlighting its great capabilities, but soon after overfitting starts to become an issue. Even though it is important that our model has the capability to learn the complex patterns presented in the data, it is equally important to find a way to generalize this learning power in order to provide better forecasting in unseen data.

Focus on Specific Regions

A limitation of our work is how we focused in what we called the "*Black Bar*" region. This was done due to great amount of information being condensed in small page-range region creating a problem that is lot more manageable to solve and can yield great results given the density of information in that area. While this approach limits greatly the model's real-world applicability, our primary focus, aligned with our thesis's motivation, was to explore the potential of image-based solutions for long-term forecasting of pages, and not to develop a fully deployable model.

6.3 Discussion and Future Direction

This thesis explores the potential of computer vision for page prefetching, prioritizing the concept itself over immediate practical application. Our work does not aim to surpass existing hardware prefetchers or *state-of-the-art* machine learning methods. The novelty of this research area in prefetching, lead to several assumptions throughout the process. These include the chosen visualization and mapping techniques, as well as the focus on *Black Bar* regions. This research significantly deepened my understanding of the topic and provided valuable insights. While the initial inexperience led to some missteps, I've gained a comprehensive grasp of the complexities involved. This newfound knowledge will allow to significantly improve any approach of future work. Additionally, the research has revealed several promising avenues for further exploration that hold the potential for significantly enhanced performance. It's important to acknowledge the significant time elapsed between the research and its documentation. The field of machine learning, particularly with the recent explosion of *Large Language Models (LLMs)* and other techniques, has undergone even more rapid growth. This opens exciting avenues for further exploration in various directions.

Exploring other visualization methods

Earlier on this thesis we emphasized on the importance of effectively transforming raw data into images for accurate predictions within image-based machine learning pipelines. Our approach primarily focuses on visualizing the time series data "in its natural state" as two-dimensional images. In these images, the x-axis represents individual timestamps, and the y-axis reflects the corresponding page range (some "compression" applied). However, the field of time series analysis offers alternative techniques for image representation, such as Gramian Angular Fields (GAF) [37, 84] and recurrence plots [23]. These methods were not implemented in our work (as explained in Chapter 3.2.2) due to the challenges associated with decomposing such images and reconstructing the original time series data from these alternate representations.

The very recently published *Drishyam* model [57] presents a groundbreaking approach to computer vision for prefetching, leveraging image classification instead of next-frame prediction. Unlike our method, which predicts the next frame, *Drishyam* formulates the problem as a classification task, aiming to prefetch data just one timestamp ahead by classifying it. By framing the problem as an image classification task, the need to convert the image back into time series data is eliminated. This opens up greater flexibility for visualizing the information, such as using the methods mentioned earlier. Building on this success, we'd like to explore similar approaches with different visualization techniques that could capture even richer information for prediction purposes and adapting in order to predict long-term in to the future.

Another very interesting aspect we didn't touch upon is the utilization of deltas. Deltas, $\Delta_N = Addr_{N+1} - Addr_N$, represent the difference between consecutive memory addresses accessed by a program. This a very common technique among prefetchers because the number of uniquely occurring deltas is often orders of magnitude smaller than uniquely acquiring addresses [33, 42, 63]. Furthermore, the power of delta information can be amplified when combined with the program counter (PC) information. The PC indicates the currently executing instruction, and together with the delta, it provides crucial context for the memory access pattern. By analyzing both PC and delta, the prefetcher can differentiate between various access patterns within a program and make more targeted prefetching decisions. This context-aware approach significantly improves the overall effectiveness of the prefetching strategy [33, 75, 57]. In our work, the chosen visualization technique limited our ability to incorporate the PC.

Exploring the use of Transformers

The recent explosion of transformer-based architectures has revolutionized various machine learning tasks. They have been originally designed for natural language processing, known for their great success in Large Language Models (*LLM's*), and recently been adapted for various computer vision tasks, leveraging their ability to model long-range dependencies and process data in parallel. This has led to significant advancements in the field, with Vision Transformers (*ViTs*) [21] being used in multiple different image based problems. Showing, also, promising results in computer vision predictions tasks [66, 31]. This thesis focused on exploring more already established methods in computer vision for our problem. However, *ViTs*, especially given their strength in keeping long range dependencies is something interesting to explore further. The *Drishyam* [57] model which was proposed a few months ago uses *SWIN Transformers* [55] another highly effective vision transformer. We must note that, if we leave behind the visualization aspect of our approach, transformers seem to offer very promising results in timeseries forecasting problems [24, 76, 87, 50] including ones with a long-term prediction [61]. An example of the above is *TransforMAP* [90] which proposes a Transformer based model for learning from the whole address space and perform multiple cache line predictions. Making it also one of the very few proposed machine learning prefetchers that predicts in the longer horizon.

Alike the work of [90, 75] we should explore developing lightweight models. These models must be optimized for minimal computational footprint, allowing for online training while a program is executing (within a running simulator). This approach would bring us closer to real-world scenarios.

Exploring Microarchitectural Simulator

Our approach to the problem followed a machine learning approach in methodology and evaluation, a crucial next step is to incorporate microarchitectural simulation for a more comprehensive assessment (ex. CMPsim [39], McPat [49] and SimpleScalar [8]). This will allow for direct comparison with established methods. Microarchitectural simulators can model the intricacies of real-world processor architectures, enabling a side-by-side comparison of our machine learning prefetcher with existing techniques. This will provide valuable insights into the relative strengths and weaknesses of each approach. Furthermore, we will be able to receive valuable performance metric measurements. Simulators facilitate the accurate measurement of critical performance metrics like *Instructions Per Cycle (IPC)* and *Execution Time*. These metrics directly reflect the impact of cache prefetching on processor performance. Analyzing these metrics will help us quantify the effectiveness of our machine learning model.

Exploring other Datasets and More Generalized Models

Due to the problem's complexity and our limited familiarity with the domain we weren't able to comprehensively and methodically evaluate the model's performance on other datasets. Looking ahead, we'd like to draw inspiration from the work presented in [78]. In that study, the authors proposed using three compact, generalized *LSTMs* instead of a specialized model for each trace type. This approach aimed to achieve better scalability. Similarly, we'd like to investigate the potential of developing a generalized CNN-based model. More specifically, CNNs are designed to work with spatial data like images, where features can appear in different locations. This allows them to recognize patterns even if they are slightly shifted or distorted. This strong ability to generalize from examples makes CNNs ideal, at least in theory, for handling complex data like ours in a more generalized model.

Bibliography

- [1] Keras documentation: Next-frame video prediction with convolutional lstms. https://keras.io/examples/vision/conv_lstm/. Accessed: 2023-08-02.
- [2] ML-based data prefetching competition. <https://sites.google.com/view/mlarchsys/isca-2021/ml-prefetching-competition?authuser=0>. Accessed: 2023-05-09.
- [3] Spec cpu 2006. <https://www.spec.org/cpu2006/>. Accessed: 2023-05-15.
- [4] Spec cpu 2017. <https://www.spec.org/cpu2017/>. Accessed: 2023-05-15.
- [5] Martín Abadi, Paul Barham, Jianmin Chen, Z. Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, 2016.
- [6] Matthew Joseph Adiletta, Farah Fargo, Mitch Diamond, Jack Adiletta, Olivier Franza, and Simon Steely. A reinforcement learning approach to optimize cache prefetcher aggressiveness at run-time. *2023 Tenth International Conference on Software Defined Systems (SDS)*, pages 95–102, 2023.
- [7] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data generating distribution, 2012.
- [8] T. Austin, E. Larson, and D. Ernst. SimpleScalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.
- [9] Mohammad Bakhshalipour, Mehran Shakerinava, Fatemeh Golshan, Ali Ansari, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. A survey on recent hardware data prefetching approaches with an emphasis on servers. *ArXiv*, abs/2009.00715, 2020.
- [10] Mohammad Bakhshalipour, Mehran Shakerinava, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Bingo spatial data prefetcher. *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 399–411, 2019.
- [11] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.
- [12] Silvio Barra, Salvatore Mario Carta, Andrea Corriga, Alessandro Sebastian Podda, and Diego Reforgiato Recupero. Deep learning and time series-to-image encoding for financial forecasting. *IEEE/CAA Journal of Automatica Sinica*, 7(3):683–692, 2020.

- [13] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite. *arXiv preprint arXiv:1508.03619*, 2015.
- [14] Peter Brockwell and Richard Davis. *An Introduction to Time Series and Forecasting*, volume 39. Springer, 01 2002.
- [15] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018.
- [16] Chris Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition*. Chapman and Hall/CRC, 03 2016.
- [17] Wei-Jie Chen, Jing-Jing Yao, and Yuanhai Shao. Volatility forecasting using deep neural network with time-series feature embedding. *Economic Research-Ekonomska Istraživanja*, 36:1–25, 06 2022.
- [18] Vincent Christlein, Lukas Spranger, Mathias Seuret, Anguelos Nicolaou, Pavel Král, and Andreas Maier. Deep generalized max pooling. In *2019 International conference on document analysis and recognition (ICDAR)*, pages 1090–1096. IEEE, 2019.
- [19] Naftali Cohen, Srijan Sood, Zhen Zeng, Tucker Balch, and Manuela Veloso. Visual time series forecasting: An image-driven approach. *arXiv preprint arXiv:2011.09052*, 2020.
- [20] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.
- [21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.
- [22] Thaleia Dimitra Doudali, Sergey Blagodurov, Abhinav Vishnu, Sudhanva Gurumurthi, and Ada Gavrilovska. Kleio: A hybrid memory page scheduler with machine intelligence. *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019.
- [23] Jean-Pierre Eckmann, Sylvie Oliffson Kamphorst, and David Ruelle. Recurrence plots of dynamical systems. *EPL*, 4:973–977, 1987.
- [24] Hajar Emami, Xuan-Hong Dang, Yousaf Shah, and Petros Zefos. Modality-aware transformer for time series forecasting. *ArXiv*, abs/2310.01232, 2023.
- [25] Mark Everingham, Luc Van Gool, Christopher K I Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.
- [26] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *ArXiv*, abs/1704.04110, 2017.
- [27] André Gensler, Janosch Henze, Bernhard Sick, and Nils Raabe. 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)*, pages 002858–002865, 2016.

- [28] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246, 2016.
- [29] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. <http://www.deeplearningbook.org>.
- [30] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. Compiling kb-sized machine learning models to tiny iot devices. *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019.
- [31] Artur Grigorev, Khaled Saleh, and Adriana-Simona Mihaita. Traffic accident risk forecasting using contextual vision transformers with static map generation and coarse-fine-coarse transformers. *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 4762–4769, 2023.
- [32] James D. Hamilton. *Time Series Analysis*. Princeton University Press, 1994.
- [33] Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns, 2018.
- [34] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.
- [35] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.
- [36] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.
- [37] Òscar Garibo i Orts, Nicolás Firbas, Laura Sebasti a, and J. Alberto Conejero. Gramian angular fields for leveraging pretrained computer vision models with anomalous diffusion trajectories. *Physical review. E*, 107 3-1:034138, 2023.
- [38] Akanksha Jain and Calvin Lin. Linearizing irregular memory accesses for improved correlated prefetching. *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 247–259, 2013.
- [39] Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob. Cmp \$ im : A pin-based on-the-fly multi-core cache simulator. 2008.
- [40] Chunhui Jiang, Guiying Li, Chao Qian, and Ke Tang. Efficient dnn neuron pruning by minimizing layer-wise nonlinear reconstruction error. In *International Joint Conference on Artificial Intelligence*, 2018.
- [41] Douglas J. Joseph and Dirk Grunwald. Prefetching using markov predictors. *Conference Proceedings. The 24th Annual International Symposium on Computer Architecture*, pages 252–263, 1997.
- [42] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, pages 364–373, 1990.
- [43] Svilen Kanev, Juan Pablo Darago, Kim M. Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David M. Brooks. Profiling a warehouse-scale

- computer. *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 158–169, 2015.
- [44] Manolis Katsaragakis, Konstantinos Stavrakakis, Dimosthenis Masouros, Lazaros Papadopoulos, and Dimitrios J. Soudris. Adjacent lstm-based page scheduling for hybrid dram/nvm memory systems. In *PARMA-DITAM*, 2023.
- [45] Jinchun Kim, Seth H. Pugsley, Paul V. Gratz, A. L. Narasimha Reddy, Chris Wilkerson, and Zeshan A. Chishti. Path confidence based lookahead prefetching. *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.
- [46] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
- [47] Pedro Lara-Benítez, Manuel Carranza-García, and José Cristóbal Riquelme Santos. An experimental review on deep learning architectures for time series forecasting. *International journal of neural systems*, page 2130001, 2020.
- [48] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [49] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–480, 2009.
- [50] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyu Zhou, Wenhui Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, 2019.
- [51] Xixi Li, Yanfei Kang, and Feng Li. Forecasting with time series imaging. *Expert Systems with Applications*, 160:113680, 2020.
- [52] Yijun Li, Sifei Liu, Jimei Yang, and Ming-Hsuan Yang. Generative face completion, 2017.
- [53] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.
- [54] Lei Liu, Shengjie Yang, Lu Peng, and Xinyu Li. Hierarchical hybrid memory management in os for tiered memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 30:2223–2236, 2019.
- [55] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.
- [56] Sarit Maitra. Take Time-Series a Level-Up with Walk-Forward Validation — sarit-maitra.medium.com. <https://sarit-maitra.medium.com/take-time-series-a-level-up-with-walk-forward-validation-217c33114f68>. [Accessed 2023-08-24].
- [57] S. Mohapatra and B. Panda. Drishyam: An image is worth a data prefetcher. In *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 51–61, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society.

- [58] Mahmood Naderan-Tahan and Hamid Sarbazi-Azad. Why does data prefetching not work for modern workloads? *Comput. J.*, 59:244–259, 2016.
- [59] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals, and Alberto Ros. Berti: an accurate local-delta data prefetcher. *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 975–991, 2022.
- [60] Kyle J. Nesbit and James E. Smith. Data cache prefetching using a global history buffer. *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 96–96, 2004.
- [61] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *ArXiv*, abs/2211.14730, 2022.
- [62] Naoya Onizawa, Sean C. Smithson, Brett H. Meyer, Warren J. Gross, and Takahiro Hanyu. In-hardware training chip based on cmos invertible logic for machine learning. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67:1541–1550, 2020.
- [63] Subbarao Palacharla and Richard E. Kessler. Evaluating stream buffers as a secondary cache replacement. *Proceedings of 21 International Symposium on Computer Architecture*, pages 24–33, 1994.
- [64] Subbarao Palacharla and Richard E. Kessler. Evaluating stream buffers as a secondary cache replacement. *Proceedings of 21 International Symposium on Computer Architecture*, pages 24–33, 1994.
- [65] Felix Petersen, Hilde Kuehne, Christian Borgelt, and Oliver Deussen. Differentiable top-k classification learning. In *International Conference on Machine Learning*, pages 17656–17668. PMLR, 2022.
- [66] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12159–12168, 2021.
- [67] Hamid Rezaatofghi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. 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)*, pages 658–666, 2019.
- [68] Pablo Romeu, Francisco Zamora-Martínez, Paloma Botella-Rocamora, and Juan Pardo. Stacked denoising auto-encoders for short-term time series forecasting. In Petia Koprinkova-Hristova, Valeri Mladenov, and Nikola K. Kasabov, editors, *Artificial Neural Networks*, pages 463–486, Cham, 2015. Springer International Publishing.
- [69] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *International Journal of Advanced Trends in Computer Science and Engineering*, 9, 10 2020.
- [70] Swapnil Sayan Saha, Sandeep Singh Sandha, and Mani B. Srivastava. Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22:21362–21390, 2022.
- [71] Hossein Sangrody, Ning Zhou, Salih Tutun, Benyamin Khorramdel, Mahdi Motalleb, and Morteza Sarailoo. Long term forecasting using machine learning methods. *2018 IEEE Power and Energy Conference at Illinois (PECI)*, pages 1–5, 2018.

- [72] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.
- [73] Manjunath Shevgoor, Sahil Koladiya, Rajeev Balasubramonian, Chris Wilkerson, Seth H. Pugsley, and Zeshan A. Chishti. Efficiently prefetching complex address patterns. *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 141–152, 2015.
- [74] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.
- [75] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '21*, page 861–873, New York, NY, USA, 2021. Association for Computing Machinery.
- [76] Ilias Siniosoglou, Konstantinos Xouveroudis, Vasileios Argyriou, Thomas D. Lagkas, Sotirios K Goudos, Konstantinos E. Psannis, and Panagiotis G. Sarigiannidis. Evaluating the effect of volatile federated timeseries on modern dnns: Attention over long/short memory. *2023 12th International Conference on Modern Circuits and Systems Technologies (MOCASST)*, pages 1–6, 2023.
- [77] Ajitesh Srivastava, Angelos Lazaris, Benjamin Brooks, Rajgopal Kannan, and Viktor K. Prasanna. Predicting memory accesses: the road to compact ml-driven prefetcher. *Proceedings of the International Symposium on Memory Systems*, 2019.
- [78] Ajitesh Srivastava, Ta-Yang Wang, Pengmiao Zhang, César A. F. De Rose, Rajgopal Kannan, and Viktor K. Prasanna. Memmap: Compact and generalizable meta-lstm models for memory access prediction. *Advances in Knowledge Discovery and Data Mining*, 12085:57 – 68, 2020.
- [79] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms, 2015.
- [80] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.
- [81] Granville Tunnicliffe Wilson. Time series analysis: Forecasting and control, 5th edition, by george e. p. box, gwilym m. jenkins, gregory c. reinsel and greta m. ljung, 2015. published by john wiley and sons inc., hoboken, new jersey, pp. 712. isbn: 978-1-118-67502-1. *Journal of Time Series Analysis*, 37:n/a–n/a, 03 2016.
- [82] Benjamin Vien, Leslie Wong, Thomas Kuen, Wing Chiu, and L.R.F. Rose. A machine learning approach for anaerobic reactor performance prediction using long short-term memory recurrent neural network. 02 2021.
- [83] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.
- [84] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. 2014.

- [85] Wei Wei, Dejun Jiang, Jin Xiong, and Mingyu Chen. Hap: Hybrid-memory-aware partition in shared last-level cache. *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 28–35, 2014.
- [86] Pinchas Weisberg and Yair Wiseman. Using 4kb page size for virtual memory is obsolete. In *2009 IEEE International Conference on Information Reuse & Integration*, pages 262–265. IEEE, 2009.
- [87] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting, 2022.
- [88] William A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23:20–24, 1995.
- [89] Pengmiao Zhang, Ajitesh Srivastava, Anant V. Nori, Rajgopal Kannan, and Viktor K. Prasanna. Fine-grained address segmentation for attention-based variable-degree prefetching. In *Proceedings of the 19th ACM International Conference on Computing Frontiers, CF '22*, page 103–112, New York, NY, USA, 2022. Association for Computing Machinery.
- [90] Pengmiao Zhang, Ajitesh Srivastava, Anant V. Nori, Rajgopal Kannan, and Viktor K. Prasanna. Transformap: Transformer for memory access prediction, 2022.
- [91] Yifei Zhang. A better autoencoder for image: Convolutional autoencoder. In *ICONIP17-DCEC*, 2018.
- [92] Yilun Zhou, Li Chen, Rong Xie, Li Song, and Wenjun Zhang. Low-precision cnn model quantization based on optimal scaling factor estimation. *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–5, 2019.