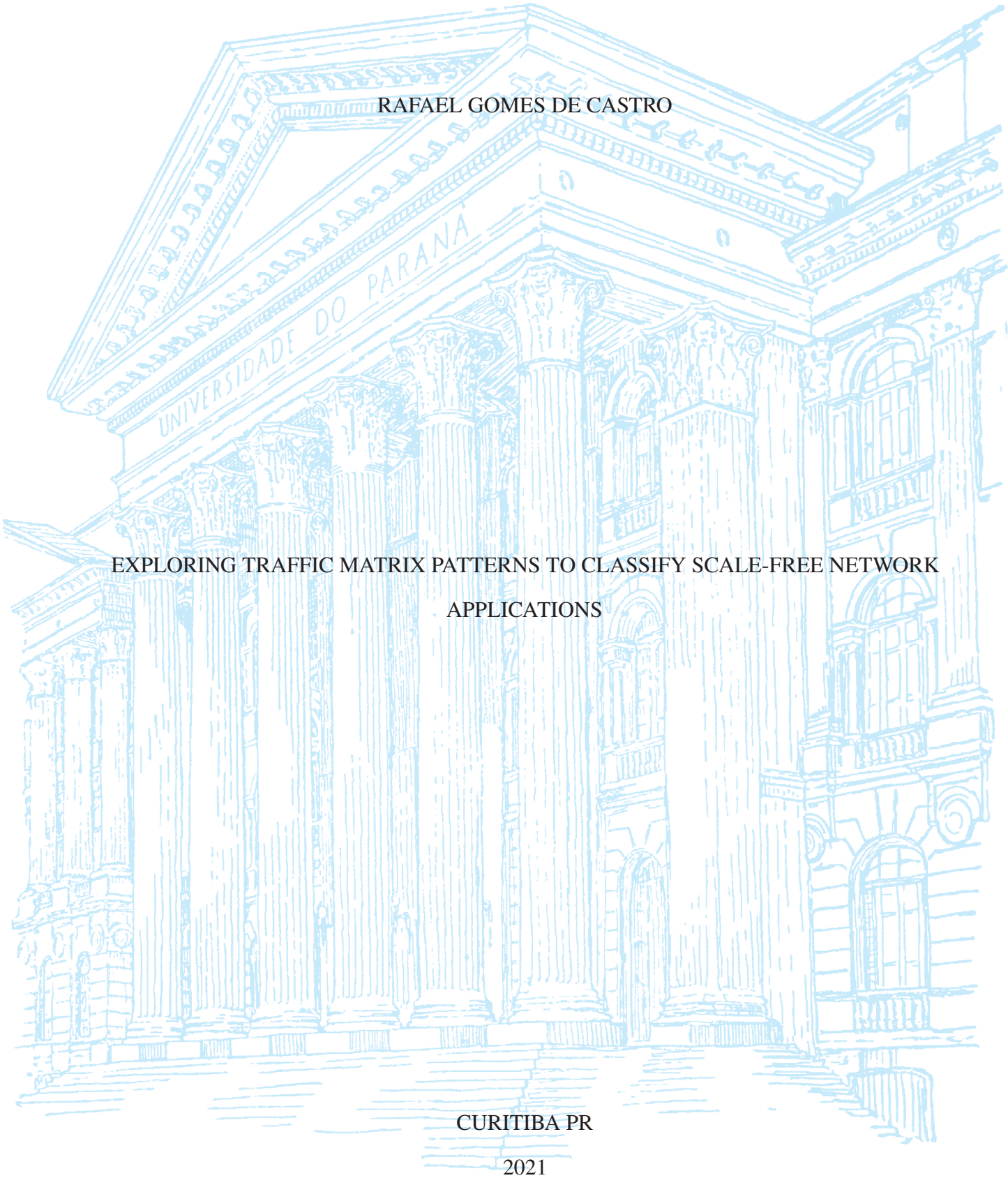
UNIVERSIDADE FEDERAL DO PARANÁ

RAFAEL GOMES DE CASTRO



EXPLORING TRAFFIC MATRIX PATTERNS TO CLASSIFY SCALE-FREE NETWORK APPLICATIONS

CURITIBA PR

2021

RAFAEL GOMES DE CASTRO

EXPLORING TRAFFIC MATRIX PATTERNS TO CLASSIFY SCALE-FREE NETWORK APPLICATIONS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Luis C. E. Bona.

Coorientador: Celio Trois.

CURITIBA PR

2021

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

C355e Castro, Rafael Gomes de
Exploring traffic matrix patterns to classify scale-free network applications [recurso eletrônico] / Rafael Gomes de Castro – Curitiba, 2021.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática.

Orientador: Luis Carlos Erpen de Bona
Coorientador: Celio Trois

1. Redes de computadores. 2. Computação de alto desempenho. I. Universidade Federal do Paraná. II. Bona, Luis Carlos Erpen de. III. Trois, Celio IV. Título.

CDD: 004.66

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **RAFAEL GOMES DE CASTRO** intitulada: **Exploring Traffic Matrix Patterns to Classify Scale-free Network Applications**, sob orientação do Prof. Dr. LUIS CARLOS ERPEN DE BONA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 23 de Agosto de 2021.

Assinatura Eletrônica
25/10/2021 11:44:30.0
LUIS CARLOS ERPEN DE BONA
Presidente da Banca Examinadora

Assinatura Eletrônica
21/10/2021 10:32:24.0
MARCO ANTONIO ZANATA ALVES
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
21/10/2021 10:36:09.0
RODOLFO DA SILVA VILLACA
Avaliador Externo (UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO)

A todos que tornaram essa trajetória possível.

AGRADECIMENTOS

Dedico essa tese em memória ao meu avô Otávio José Gomes, que veio a falecer dias após a minha defesa. Agradeço todo o seu esforço por me ensinar a importância do estudo, pois eu não teria chegado à titulação de mestre sem os seus ensinamentos, bem como aos seus ensinamentos sobre a importância da família. Sua humildade e lições serão legados que me acompanharão até o último dos meus dias. Serei eternamente grato por ter compartilhado ao seu lado quase 28 anos da minha existência.

Agradeço também à minha mãe Maria Alice Gomes de Castro e à minha avó Bernardina Furtado Gomes, que, juntamente com o meu avô, enfrentaram a árdua tarefa de educar uma criança, e se mantiveram ao lado dela em todos os momentos. Estou encerrando mais um ciclo da minha vida onde a participação dessas duas mulheres foi indispensável. Devo tudo a elas.

Um saudoso muito obrigado aos meus amigos Jean Carlo Kurpel Diogo e Fabrício José de Oliveira Ceschin, que me ajudaram com o estudo dos algoritmos utilizados na minha tese; e aos meus amigos Danielle de Fátima Ivanchechen Diego Hirt Santos Rodrigues e Vinícius Fülber Garcia, cuja caminhada durante a graduação e mestrado tornou-se mais fácil com a parceria deles. Agradeço também aos meus colegas Egon Hilgenstieler, Juliana Matsumoto, Luiz Bettoni, Débora Sandi e Eliane Cajola, que trouxeram palavras de apoio em momentos difíceis, me ajudaram a formular planos para superá-los, e a ensaiar a apresentação para defesa. E a todos os meus amigos que formei durante a minha trajetória acadêmica, a gratidão à vocês será para sempre.

RESUMO

A evolução da computação e das redes de computadores permitiu a interconexão de vários computadores, agregando seus poderes de processamento para formar arquiteturas de computação de alto desempenho (HPC). As aplicações executadas nesses ambientes computacionais processam e comunicam grandes quantidades de informações, levando várias horas ou até dias para concluir suas execuções. Portanto, entender suas demandas de computação e comunicação é essencial para fins de gerenciamento. Embora a maioria das aplicações HPC sejam implementadas com algoritmos conhecidos que tendem a seguir um determinado padrão em computação e comunicação, os métodos clássicos de análise de tráfego não são precisos para classificá-los. Nesse sentido, argumentamos que observar e entender os padrões visuais nas matrizes de tráfego (TMs) dessas aplicações pode fornecer um método de classificação preciso. Neste trabalho, propomos o SCTReco (**S**cale **F**ree **T**raffic matrices **Reco**gnition), um framework que mantém um banco de dados com caraterísticas visuais extraídos dessas TMs e aplica técnicas de aprendizado de máquina e aprendizado profundo para classificar as aplicações HPC que estão consumindo a rede, independentemente do número de nós computacionais que a executam. O modo de classificaçao do SCTReco usa algoritmos de pré-processamento, extratores de características e classificadores de Aprendizado de Máquina (ML). Equipamos o framework com os conhecidos classificadores Random Forest (RF) e Support Vector Machine (SVM); como extrator de características são usados o Padrão Binário Local Uniforme (ULBP), Padrão Binário Local Robusto (RLBP), Matriz de Coocorrência em Nível de Cinza (GLCM) e Análise de Componente Principal (PCA); e pré-processando as TMs com interpolação do vizinho mais próximo, desfoque gaussiano, recorte e dilatação. Com essa abordagem, SCTReco pôde ser treinado com TMs de 128 nós e reconheceu três aplicativos HPC reais executados em 256 nós de computação com 99,76% de precisão e 85,23% reconhecendo TMs de 512 nós de computação.

Palavras-chave: Redes Definidas por Software, Padrões de Comunicação, Aplicações HPC

ABSTRACT

The evolution of computing and computer networking allowed multiple computers to be interconnected, aggregating their processing powers to form High-Performance Computing (HPC) architectures. Applications running in these computational environments process and communicate huge amounts of information, taking several hours or even days to complete their executions. So, understanding their computation and communication demands is essential for management purposes. Although most HPC applications are implemented with known algorithms that tend to follow a certain pattern in computing and communication, classical traffic analysis methods are not accurate to classify them. In this context, we argue that observing and understanding the visual patterns in the traffic matrices (TMs) of these applications can provide an accurate classification method. In this work, we propose SCTReco (**S**cale **F**ree **T**raffic matrices **Reco**gnition), a framework that maintains a database with visual characteristics extracted from these TMs and applies machine learning techniques to classify the HPC applications that are consuming the network, regardless of the number of computational nodes that execute it. SCTReco's classification mode uses preprocessing algorithms, feature extractors, and Machine Learning (ML) classifiers. We equipped the framework with the well-known classifiers Random Forest (RF) and Support Vector Machine (SVM); using as feature extractor the Uniform Local Binary Pattern (ULBP), Robust Local Binary Pattern (RLBP), Gray-Level Co-occurrence Matrix (GLCM), and Principal Component Analysis (PCA); and preprocessing the TMs with nearest neighbor interpolation, Gaussian blur, crop and dilation. With this approach, SCTReco could be trained with TMs of 128 nodes and recognized three real HPC applications executed in 256 computing nodes with 99.76% of accuracy, and 85.23% recognizing TMs of 512 computing nodes.

Keywords: Software Defined Network, Communication Patterns, HPC Applications

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

API	Application Program Interface
CNN	Convolutional Neural Network
DPI	Deep Packet Inspection
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
HPC	High-Performance Computing
IDS	Intrusion Detection System
IP	Internet Protocol
LBP	Local Binary Pattern
ML	Machine Learning
MPI	Message Passing Interface
NPB	NAS Parallel Benchmarks
PPGINF	Programa de Pós-Graduação em Informática
QoS	Quality of Service
RF	Random Forest
RFB	Radial Basis Function
RLBP	Robust LBP
SDN	Software-Defined Networking
SVM	Support Vector Machines
TIM	Topology Information Module
TM	Traffic Matrix
UFPR	Universidade Federal do Paraná
ULBP	Uniform LBP
VM	Virtual Machine

SUMÁRIO

1 INTRODUCTION

High-Performance Computing (HPC) has become one of the most important methods to do scientific research. In the current society, HPC has an important influence on National strategy. At present, HPC has been used in the fields of aerospace, automotive, nuclear simulation, weather forecast, and so on. People can do many experiments that normal computers are not capable of by HPC, so that huge real experiments fees can be saved and there will be no bad influence on the environment (Bo et al., 2012).

However, resource management is a hard problem, due to the scale of modern data centers, the heterogeneity of resource types and their interdependencies, the variability and unpredictability of the load, and the range of objectives of the different actors (Jennings e Stadler, 2015). Furthermore, the ideal solution has to combine performance, cost, and resilience too. Considering these requirements, Software Defined Network (SDN) has several features that make it attractive for HPC (Lee et al., 2016).

SDN separates the control's logic from the data forwarding devices. In traditional networks, the network administrators have the difficult task to reconfigure each network device individually any time a new configuration is needed. Without this vertical integration, networks built over the SDN paradigm can be configured on the fly, bringing to the network the dynamism necessary to HPC applications (Nunes et al., 2014).

Identifying and categorizing network traffic flows are fundamental to a variety of networking applications including management, Quality of Service (QoS), and security. In addition, such flow-level classification needs to be effective and efficient. The methodologies of flow-level classification range from protocol port identification, deep packet inspection, and, more recently, Machine Learning (ML) approaches. ML-based methods have undergone extensive investigation, in particular, due to the competitive detection accuracy (He et al., 2016).

ML-based algorithms simply require obtaining a set of flow features, obtained from a switch when the traffic passes through (He et al., 2016). Using large data set and calculate features, ML-based algorithms can construct robust classification models. Moreover, the statistical properties-based features of the network traffic are also becoming important for ML-based classifications such as source address and destination address (Spatscheck et al., 2014).

The majority of HPC applications are implemented using well-known patterns. These patterns are known in the literature as dwarfs (Asanovic et al., 2006). In each dwarf, there is an essentially similar amount of information transmitting among a set of computational nodes. Although HPC applications tend to follow a "well-behaved" communication pattern, the classical methods of traffic analysis, such as port-based, Deep Packet Inspection (DPI), or even using ML with a multiple and diverse vector of features (Eerman et al., 2006; Fahad et al., 2014b; Soysal e Schmidt, 2010b; Zhang et al., 2012) are not accurate for classifying them.

In the literature, very few work explored the communication patterns of HPC applications, extracting and inspecting their Traffic Matrices (TMs) as a way to capture their communication patterns. In (Trois, 2017), the authors introduced a novel method that renders different visual textures for each applications' communication patterns, named TReco (**T**raffic matrix **Reco**gnition)[1]. It relies on two well-known textural representations, Uniform Local Binary Pattern (ULBP) (Ojala et al., 2002) and Robust Local Binary Pattern (RLBP) (Zhao et al., 2013a) for extracting the feature vectors of the TMs. Despite the classification accuracy has been over

[1]TReco is open-source, publicly available at: www.inf.ufpr.br/ctrois/treco/

99%, there was a limitation for keeping this high accuracy, requiring that the TMs for training must be the same size as for testing.

In realistic scenarios, clusters are often partially allocated to applications, i.e. the application can typically run at different scales varying drastically the number of communicating nodes, depending on the size of the problem or resource availability. So, now we propose a framework for classifying HPC applications based on TRECO that is TM textural centric.

SCTRECO (**S**cale **F**ree **T**raffic matrices **Reco**gnition) is grounded in (Trois, 2017). Traffic matrices indicates what is used as knowledge base for applications recognition. Scale-free indicates that the framework is able to recognize applications without concern of them scales. In our tests, we explored the traffic matrices with different scales to classify the network applications.

Since TRECO already combines feature extractors and classifiers able to recognize TMs with the same scale, SCTRECO focus mainly on making the TMs as similar as possible before feature extraction and classification. To do so, we separately evaluated different preprocessing strategies: interpolation, blur, crop, and dilation, aiming to identify the one that causes the highest recognition rate in a dimenison-free scenario.

Beyond the TRECO's feature extractors ULBP and RLBP, we explored the feature extractors Gray Level Co-occurrence Matrix (GLCM), and Principal Component Analysis (PCA). SCTRECO maintains the TRECO's classifiers Random Forest (RF) and Support Vector Machine (SVM). In our evaluations, we noticed that SCTRECO recognized with 85.23% of accuracy the TMs with scale 4x bigger than that used in training step. This recognition rate was achieved training SCTRECO with 128x128 TMs and testing 512x512 TMs. Using RLBP and SVM, SCTRECO reached an accuracy value of 99.76% recognizing three applications running in HPC infrastructures. This accuracy value was reached using 128x128 TMs for training and 256x256 TMs for testing.

The organization of the rest of this paper is presented below. Chapter 2 explains HPC and SDN, describing the main advantages of the usage of this network paradigm as infrastructure to build an HPC when compared with traditional networks. Chapter 3 focus on the communication patterns, explaining the computational dwarfs described in (Asanovic et al., 2006) and also shows the classification strategies used in our work. Chapter 4 explains our proposed framework SCTRECO to deal with the limitation of TRECO, and contains the results obtained by SCTRECO. Finally, Chapter 5 concludes our work.

2 COMMUNICATION PATTERNS IN HIGH PERFORMANCE COMPUTING AND SOFTWARE DEFINED NETWORKS

In High-Performance Computing (HPC) applications, the interconnection bandwidth among computing nodes is essential. Software Defined Network (SDN) allows the identification and programmatically treatment of each network flow, using a predefined set of rules, without the intervention of the network operator, that is activated by the flow of specific traffic. Coupling this paradigm with High Performance Computing (HPC) allows a tighter integration of network resource and computational power, bringing to HPC the ideal combination of *performance*, *cost*, *scalability*, and *resilience* (Lee et al., 2018; Lee et al., 2016).

This chapter contains five sections. In Section 2.1, we describe the main issues found in traditional HPC clusters and common strategies found in the literature to solve them. In Section 2.3, we explain the concepts of SDN, including its architecture, protocols, and operation mode. The benefits of SDN to HPC clusters are explained in Section 2.3.1. The Section 2.2 describes the communications patterns found in HPC applications. Finally, the Section 2.4 concludes the chapter.

2.1 HIGH PERFORMANCE COMPUTING

In the last few decades, scientists use HPC applications to create and predict complex phenoms, for example, weather forecasting, prediction of natural disasters, bacterial profiling, animal genotyping, and so forth (Gupta e Milojicic, 2011). Due to the considerable time that HPC applications take for finishing their executions, many researchers are studying and presenting proposals for accelerating them. The application's performance may be affected by several aspects, creating multiple study opportunities and making this area widely researched. There are works modifying scheduler for optimizing the job placement (Renner et al., 2015; Al-Fares et al., 2010; Lee et al., 2014), using FPGA (Dimond et al., 2011; Vassiliev, 2017) or GPU (Erlacher et al., 2017; Tangherloni et al., 2017), proposing improved programming languages and frameworks (Nugteren, 2017; Kuster, 2017), employing cache (Yu et al., 2017; Gémieux et al., 2017), taking advantage of cloud resources (Righi et al., 2016; Galante et al., 2016), and so forth.

Another aspect used for improving the HPC applications is the fact that the wide majority is implemented using well-known numerical methods (Asanovic et al., 2006). A computational dwarf can be defined as "a pattern of communication and computation common across a set of applications". These communication and computation patterns are also intensely researched and have been used for optimizing the communication on networks-on-chip (Werner et al., 2017), GPU (Wang et al., 2016), multiprocessor architectures (Prabhakar et al., 2017), and ameliorating the performance of applications (Rubin et al., 2014). Section 2.2 provides more details about the communication patterns found in HPC applications.

The network system is also widely studied for improving the performance of these applications. The main reason is that the network performance did not evolve at the same rate that other HPC resources, becoming a bottleneck. For example, comparing the network transmission rate with the individual node computational power, in last decade, the computational power grew 36 times more than the network bandwidth (Jain, 2016). So, understanding the communication demands of HPC applications and classifying them is a challenge in networking research (Srivastava et al., 2016) because it can be instrumental in several management activities,

such as monitoring (Chowdhury et al., 2014; Van Adrichem et al., 2014; Yu et al., 2014), expansion planning (Simmons, 2014), traffic engineering (Trestian et al., 2013; Wang et al., 2008; Akyildiz et al., 2014), detection of anomalies (Giotis et al., 2014), and energy saving (Yao et al., 2016).

Trying to improve current HPC routing solutions, avoiding congestion in the network, Samuel et al. (2017) introduced a scalable routing paradigm for HPC networks that decouples intra- and inter-application flow contention named *Routing Key*. *Routing Key* contains two main algorithms: *Application Routing Keys* (ARK) and *Network Key Routing* (NKR). The ARK algorithm proactively allows each self-aware application to route its flows according to a predetermined routing key. In NRK algorithm, a centralized scheduler chooses between several routing keys for the communication phases of each application, and therefore reduces inter-application contention while maintaining intra-application contention-free routing and avoiding scalability issues.

Each application is then associated with this set of different routing keys. Using both ARK and NRK, Samuel et al. improved the communication runtime by up to 2.7x, but their work considers only HPC performance issues. To bring more performance, dynamism, and ease of configuration to HPC clusters, the idea of HPC infrastructures constructed over programmable networks (SDN) has been proposed.

Optimizing the communication of HPC applications is also challenging due to the limited control over the system environment. Unlike the computation resources that are typically known a priori and are fully under the control of an executing application, the availability of network resources may neither be predictable nor be completely under the control of an executing application (Jain, 2016). The common assumption is that HPC applications run on a fixed number of processes, where the network is considered a static resource, working as a connectivity service that cannot be controlled or modified (Righi et al., 2016).

Moreover, the current network stack provides a best-effort service model for hosts communication, which is not enough to satisfy the on-demand data access required by these applications (Sakr et al., 2011). For combating these problems, SDN emerged as an architecture decoupling the network control and forwarding functions, enabling the network to become directly programmable, and the underlying infrastructure being abstracted for applications and services (ONF, 2013). One benefit offered by SDN is that the network can be modified according to the user requirements, inclusive for suiting transient demands, allowing runtime adjustments for improving the performance of specific applications.

In SDN, forwarding hardware is decoupled from control decisions, with this logically centralized as a software-based controller. There is a need for SDN-based HPC solution for enhanced throughput and better utilization of bandwidth for meeting the need of today's Scientific and Business computation needs (Krishna et al., 2017). Section 2.3 provides more details about the main concepts, protocols, operation mode, and benefits of SDN. In the next section, we introduce the communications patterns found in HPC applications.

2.2 COMMUNICATION PATTERNS IN HPC

As HPC applications operate with large volumes of data that flows across the computation nodes, any minimal improvement in the communication performance can result in a much better performance at the execution time. Understanding the application pattern is a great step to achieve better knowledge about the optimal network configuration and thus reach better performance when running this application.

In 2003, Vetter and Mueller develop a study about the communications patterns present in several scientific applications that were extracted from benchmarks for large cluster architecture (Vetter e Mueller, 2002). Through Message Passing Interface (MPI), they analyzed inherent characteristics present in each application: point-to-point and collective communication, revealing several similarities between them. In point-to-point communication, they measured the number of messages, type, payload size, and destination. For collective communication, they determined the type, frequency, and payload size.

Colella made contributions beyond traffic pattern identification using some packet header. Colella identified seven patterns in important numerical methods for science and engineering, naming them as seven dwarfs (a Snow White and the Seven Dwarfs satire). He formally called them Dense Linear Algebra, Sparse Linear Algebra, Fast Fourier Transform, Structured Grids, Unstructured Grids, Particles, and Monte Carlo. These methods constitute classes defined by similarity in computation and data movement (Colella, 2004). Asanovic et al. updated Colella's list with six more methods and renamed three already known methods. Fast Fourier Transform was renamed to Spectral Methods, Particles was renamed to N-Body Methods and Monte Carlo came to be called MapReduce. The new dwarfs are Combinational Logic, Graph Transversal, Dynamic Programming, Backtrack & Branch+Bound. The importance of Colella's and Asanovic's works lies in the fact that distributed parallel programs fall into at least one of these 13 dwarf classes of communication patterns. In addition, if the variance of the expressed patterns is bounded, identification of the dwarf class should be possible solely from observed communications. An interesting approach to be considered when analyzing the communication patterns is the measurement of the traffic matrix (TM). A TM represents the amount of data traffic between origin and destination in the network. TMs can be useful in many network engineering applications, such as network survivability analysis, traffic engineering, and capacity planning. With the use of TMs, spatial and temporal behavior can be analyzed.

Spatial behavior can be formalized as a traffic matrix M_B, where each position $M_B[i][j]$ holds the number of bytes transmitted from node i to node j during a specific period of time. The matrix was normalized to its maximum value for graphical visualization purposes. For graphical visualization the images are representing in grayscale: black cells are the communicating pair of nodes and white cells indicate that no communication happened. Figure 2.1 gives the spatial behavior of selected applications, showing the total amount of data exchanged throughout their execution time. These applications are a set of benchmarks developed by NASA Advanced Supercomputing Division, called NAS Parallel Benchmarks (NPB). From these benchmarks, those that most exchanged information among computation nodes are bt, cg, ft, and lu. It is possible to see that these applications feature different spatial communication behavior. *ft* exchanges almost the same amount of data across all nodes, transmitting a maximum of 185.7 MB by a pair of nodes. *cg* is the program that most exchanged data, considering the pair of nodes (597.1 MB), however, as we can see in this spatial behavior matrix, only a few pairs of nodes communicated. At least, the total amount of data transmitted from all computation nodes are also displayed (Trois et al., 2016).

Temporal behavior includes the notion of time to M_B. Formalizing, $M_T[t](M_B[i][j])$ describes that each instance of t holds a spatial behavior matrix $(M_B[i][j])_{t''}$, filled with the amount of data transmitted during the interval [t" - t'], where t' is the instance of time just before t". Figure 2.2 shows a graphical representation of M_T for *ft*. At the moment $t(1)$, it is initializing, and there is an intensive communication among the master node $i8$ with all other nodes. In $t(2)$, the node $i1$ starts data transmission to $i2$ and, in the following moments, to $i3$, $i4$, $i5$, successively (Trois, 2017).

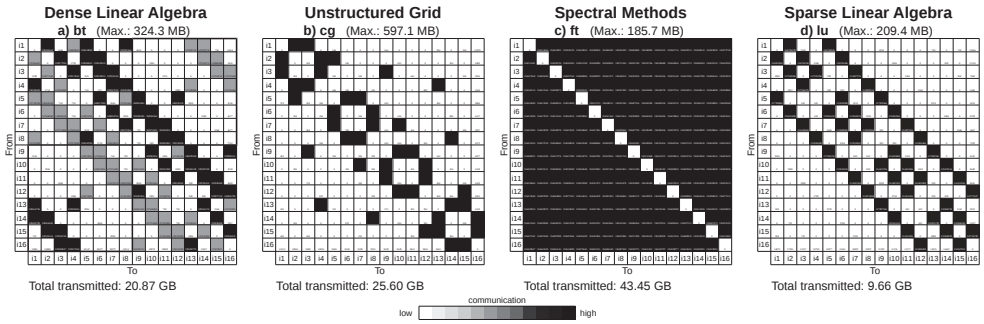


Figura 2.1: Spatial behavior through traffic matrices

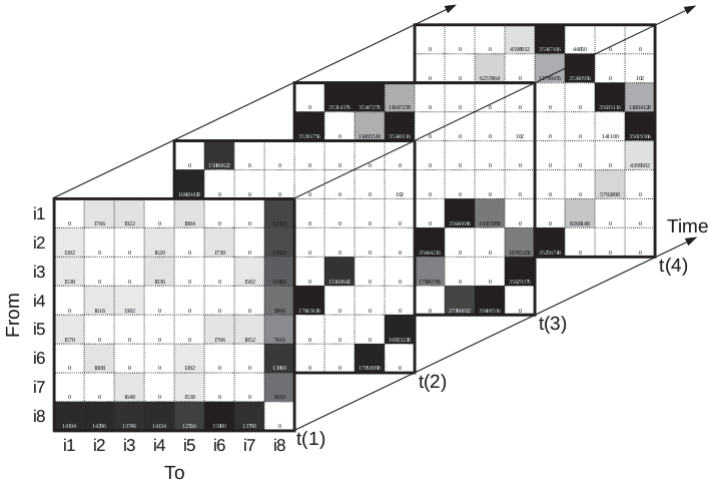


Figura 2.2: 3-D temporal behavior representation

Another interesting aspect that TMs can graphically show us is the pattern presented in the application communications. Regardless of the number of the computation nodes or the input data, it is possible to visually identify the communication patterns on their TMs. Figure 2.3 shows the TMs generated when executing *lu* application in HPCs with different number of computing nodes (8, 16 and 25 nodes).

To automatically set the network with the best configuration for the running HPC application, the first challenge is to detect which application is in execution. Existing proposals for detecting the communications can be inserted in one of the three groups, where no one of these three groups can be considered a perfect solution. The first approach uses a port-based classification process, a strategy known to be inaccurate and leading many applications to adopt dynamic port numbering to overcome the performance limitations of networks. The second approach uses deep packet inspection (DPI). This strategy examines the payloads of the packets for classifying traffic, demanding high computational power, and is impossible to be done without specific knowledge about the application protocols. Furthermore, there are applications that use cryptographic methods to ensure that only the destination node will be able to read the message, making DPI impossible. The last approach uses ML algorithms for traffic classification. In this method, intrinsic and statistical flow are exploited as representations for feeding ML classifiers. Packet size average and variance, the total number of packets bytes, flow duration, server, and client port numbers are examples of packet information used on works involving ML in traffic classification. Most of these works reach high accuracy.

Several works were published aiming to classify applications according to dwarfs computing methods. Rodinia (Che et al., 2009) is an open-source benchmark suite that

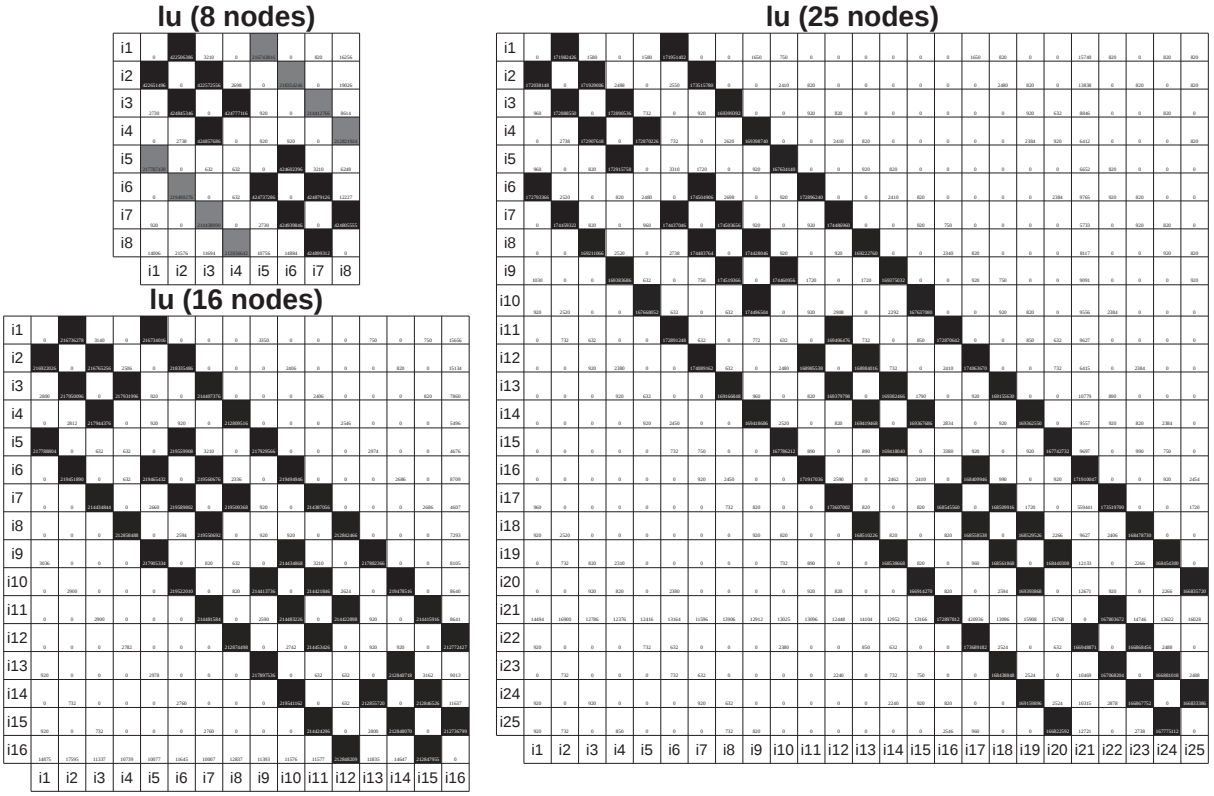


Figura 2.3: Communication behavior varying the amount of computing nodes

implements a wide range of applications with different computation and communication patterns. The implemented applications can be mapped to a subset of the 13 Dwarfs. Rodinia uses OpenMP and CUDA in order to compare multicore CPUs vs. manycore GPUs. Similar to Rondinia, Springer (2011) proposed the use of Dwarfs focused on GPU implementations. They discuss three benchmark suites that implement a subset of the 13 Dwarfs on the GPU, list typical problems related to efficient GPU implementations, and discuss the specific problems and performance with respect to some GPU Dwarfs.

OpenDwarfs is a high-performance benchmark computing suite for heterogeneous architectures (Feng et al., 2012). This suite offers benchmarks that are classified by thirteen dwarf categories, with implementation in OpenCL, a vendor-agnostic and open-standard computing language for parallel computing. OpenDwarfs has an advantage over Rodinia, being this the most mature suite, but that covers all possible dwarfs. In the next section, we present the main concepts, protocols, operation mode, and benefits of SDN.

2.3 SOFTWARE DEFINED NETWORKS

Traditional networks are vertically integrated, with the control plane (that decides how to handle network traffic) and data plane (that forwards traffic according to decisions made by the control plane) bundled together inside each network device (Kreutz et al., 2015). This vertical integration imposes the difficult task for network administrators to reconfigure each network device individually any time a new configuration is needed. As typical networks have numerous switches, routers, and other forwarding devices, network management and performance tuning are challenging and thus error-prone (Nunes et al., 2014).

Breaking the vertical integration by separating control's logic from the underlying routers and switches that forward the traffic, SDN (Software-Defined Networking) brings to the

network a centralized logical control that simplifies the reconfiguration of the network. With the separation of control and data planes, network switches become simple forwarding devices (Nunes et al., 2014). With a centralized control plane in a single device, the controller has a view of the entire network which enables a better balancing of arriving flows, allows the computation nodes being relocated according to their communications patterns and also allows the switches' flow tables to be rearranged for optimizing the forwarding (Kreutz et al., 2015).

As shown in Figure 2.4, SDN architecture is composed of a control plane/controller and a data plane. The controller provides an abstract view of the entire network infrastructure. The most common controller used in SDN is the NOX controller. The data plane contains all the forwarding devices. The correct communications between the network controller(s) and data devices require protocols interpreted by them. SDN protocols are classified into "regions". If a network has more than one controller (multi-controller-based architecture), East-West protocols guarantee communication between the various controllers. Northbound application programming interfaces (APIs) represent the software interface between the software modules of the controller platform and the SDN applications running atop the network platform. Named as Southbound protocols, OpenFlow is the most popular SDN protocol that provides the information exchange between the controller(s) and the data devices.

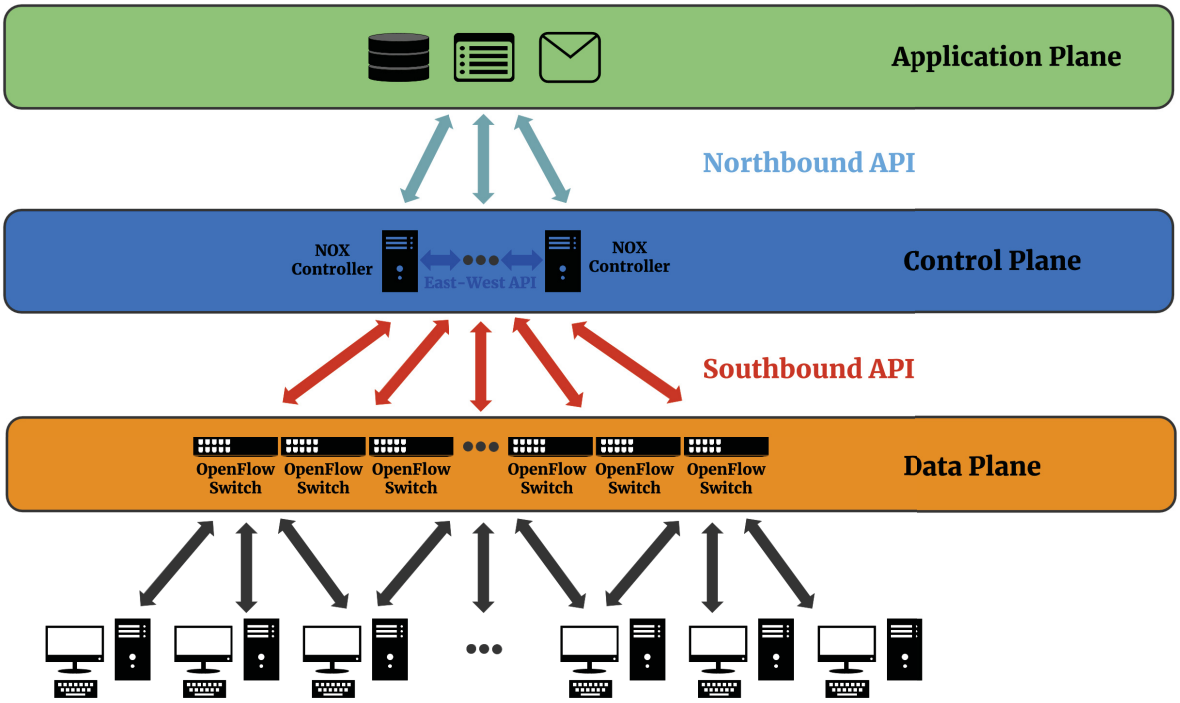


Figura 2.4: SDN architecture

An OpenFlow switch is composed of flow and group tables. Each flow table contains one or more flow entries, specific to a particular flow and is used to perform lookup and forwarding. The OpenFlow controller is able to manage these flow entries throw messages exchange between the switch and the controller. With a single look-up on the flow table, the switch can perform the desired forwarding for each incoming packet. The flow entries are sequentially numbered on the flow table, starting by 0. For every incoming packet, the switch runs over each flow entry presented on the flow table, and, founding one rule that matches with the packet, the corresponding forwarding is executed. An important point is that flow entries may also contain a group. A group table consisting of group entries offers additional methods of forwarding (i.e.

broadcast, multicast, and link aggregation). A group entry is made by a group identifier, a group type, counters, and a list of action buckets (each action contains a set of actions to be executed and associated parameters).

OpenFlow also provides a basic set of management tools, useful to control features such as topology changes and packet filtering. This protocol basically consists of many OpenFlow-enabled switches and one or more OpenFlow controllers. An OpenFlow switch contains one or more flow tables and a group table and performs packet lookups and forwarding. The OpenFlow controller is responsible for the distribution of appropriate instructions to the data devices, in order to maintain all the network protocols and policies.

2.3.1 HPC with SDN

SDN functionality provides ample opportunities both at the application level and the system level to optimize for HPC workloads by providing custom adaption for the traffic patterns. With SDN, the HPC systems can reach an ideal combination of *performance*, *cost*, *scalability*, and *resilience* (Lee et al., 2016).

- Performance: SDN allows for dynamic reconfiguration of the network to provide per-flow resource management and routing, which is significantly more flexible than the deterministic routing scheme that is widely employed in the current HPC clusters. The ability to manage traffic at the flow level potentially enables network resources to be utilized much more effectively.

- Cost: SDN is designed for Internet and data center applications with large numbers of computing nodes. The economics of scale dictates that SDN technology will be more cost-effective as the technology matures.

- Scalability: The network operations in an SDN are simpler than those in networks with advanced adaptive routing schemes such as the global adaptive routing in the Cray Cascade system. Hence, SDN is more scalable than interconnects with advanced adaptive routing schemes and may strike the ideal balance between the network complexity and capability for future HPC systems.

- Resilience: The flexible system reconfiguration in a SDN facilitates resilience management at the network level, which has become increasingly important as the system size increases.

The Message Passing Interface (MPI) library is a de facto standard for developing HPC application (Achour e Nasri, 2012). Takahashi et al. (2015) implemented some SDN enhanced MPI primitives; they developed an MPI *Bcast* (broadcast) for eliminating duplicate packets in the network and an MPI *Allreduce* that makes a communication plan to forward the reduction through distinct paths. Date et al. (2015) discussed the integration of SDN with HPC infrastructure, extending Takahashi et al. (2015) work by incorporating a network-aware HPC job placement. They also proposed using SDN for modifying the forwarding on network failures.

Polezhaev et al. (2014) proposed two modifications on BackFill job placement algorithm (Feitelson e Weil, 1998) for making it network-aware. They also used SDN for modifying the network to improve inter-job communication. Similarly, Jamalian e Rajaei (2015) proposed A SDN Empowered Task Scheduling System (ASETS) that allows to schedule data-intensive HPC tasks in a Cloud environment. They used the “bandwidth awareness” capability of SDN to better use the bandwidths when assigning tasks to virtual machines.

Alsmadi et al. (2016) proposed SDN-HPC, a model that allows executing multiple HPC jobs simultaneously by creating multiple network slices. Their approach observes the hosts that are exchanging high volumes of traffic and avoids placing new HPC jobs on these hosts. Finally, Wu et al. (2016) implemented a network congestion detector based on the traffic monitoring mechanism using SDN. They also designed a traffic flow scheduler that is capable of dynamically redistributing the network traffics for fat-trees rerouting congested paths to the congestion-free ones.

With these advantages, HPC systems and applications can take advantage of SDN features to maximize their effectiveness. Instead, using SDN, the network can be configured on the fly to operate with the best performance for the running application (Kreutz et al., 2015), without the intervention of the network operator, avoiding human errors in the reconfiguration. So, in our work, we focus on identify which HPC application is running on SDN infrastructure. One strategy to identify the running application in an HPC cluster is observing its communication pattern.

2.4 CHAPTER REMARKS

In this chapter, we introduced the concepts and main issues found in traditional HPC clusters. We also saw some efforts for characterizing the communication patterns through some information such as message type, destination, frequency distributions, and size. We presented the concept of dwarfs for classifying the communication patterns expressed by HPC applications. We related their communication requirements, mapping the application areas and exposing the importance of dwarfs in several computational areas.

We investigated how the communication patterns can be expressed through traffic matrices, considering their spatial and temporal behaviors. We concluded that these applications tend to transmit the same amount of data across the same computing nodes and we call this as a “well-behaved” communication pattern. Computational approaches for classifying the communications were also described in this chapter.

Lastly, we described the SDN topology and the advantages compared with traditional networks. Following, we saw how SDN can optimize HPC workloads. Despite the diversity of published works using SDN for optimizing applications, to the best of our knowledge, Trois (2017) was the first one to exploit the well-behaved communication patterns expressed by the HPC applications for reprogramming the network, aiming to accelerate the applications. In the next chapter, we present this first attempt of HPC applications optimization using the communication patterns, and the algorithms used in our work to improve it.

3 APPLICATION CLASSIFICATION PROCESSING

As HPC applications present well-known communication patterns and the SDN offers flexibility for modifying the network dynamically, we can develop a high-level abstraction using the communication patterns for programming the network, aiming to speed up HPC applications. Trois et al. started the development of this high-level abstraction, developing a TM-based framework, named TRECO, that uses ML approaches to identify the running HPC application (Trois et al., 2018).

Although the innovative approach, TRECO only can recognize applications when the TMs used in the training phase are the same as that used in the test phase. Considering that HPC clusters can have n number of computing nodes, TRECO needs to be upgraded to make it viable. Now, we are proposing SCTRx, a framework able to classify TMs even when trained with TMs of another scale.

The general idea behind this work is handle the TMs as images and classify them using image classification strategies. So, we used ML approach to recognize TMs, applying preprocessing algorithms and feature extraction to improve the recognition rate.

This chapter describes the preprocessing, feature extraction and recognition algorithms used in our work. Section 3.1 describes the preprocessing algorithms; Section 3.2 present the feature extractors; Section 3.3 shows the ML algorithms; and finally Section 3.4 present the framework that precedes our approach on TMs recognition.

3.1 PREPROCESSING ALGORITHMS

The purpose of image preprocessing is to eliminate irrelevant information in the image, restore useful real information, enhance the detectability of the relevant information, and simplify the data to the maximum, thereby improving the reliability of image feature extraction and recognition (Zhang et al., 2020). One common method is weighted average grayscale, which consists of transforming a RGB image into a one-channel image using the following function to do it:

$$Gray(i, j) = 0.299 * R(i, j) + 0.578 * G(i, j) + 0.114 * B(i, j), \tag{3.1}$$

where *R(i,j)*, *G(i,j)* and *B(i,j)* are the value of red channel, green channel and blue channel, respectively, in the pixel *(i,j)* of the image.

Beyond weighted average grayscale, our work is applied to other preprocessing algorithms, presented in this chapter. First, we will explain the blur algorithm, next we will explain the interpolation, and finally, we will show the dilation algorithm.

3.1.1 Blur

A blurred image can be expressed as a matrix B, where

$$B = K \times S + N \tag{3.2}$$

K, S and N are matrices respectively representing the blur kernel, the latent sharp image, and additional noise. The $\times$ denotes convolution (Koh et al., 2021).

In Vasiljevic et al. (2017), the authors studied the effects of using blurry images as input to algorithms for recognition. They found that there was a fair bit of generalization across certain blur types, but there was a limited generalization from radially symmetric defocus blur to oriented one-scalabe motion kernel. While precise knowledge of the blur in an image is helpful, recognition with unknown blur can be made almost as robust by using models trained with a diverse set of kernels. With different types of blur on training images, the models can computing blur-invariant features in their hidden layers.

3.1.2 Interpolation

Image interpolation is a term often used with different terminologies in literature, like image scaling, image resampling and image resize (Parsania e Virparia, 2016). Interpolation techniques determine the values of a function at positions lying between its samples. One common interpolation technique is called nearest neighbor, and its the simplest interpolation. In this method, each interpolated output pixel is assigned the value of the nearest sample point in the input image. The interpolation kernel for nearest neighbor

$$h(x) = \begin{cases} 0, \text{|x|} > 0 \\ 1, \text{|x|} < 0 \end{cases} \tag{3.3}$$

The frequency of the nearest neighbor kernel is

$$H(\omega) = sinc(\omega/2) \tag{3.4}$$

In Sarwinda et al. (2021), the authors used ResNet-18 and ResNet-50 architecture to classify colorectal cancer images into benign and malignant. The authors resized the fundus image into a 224 x 224 grid. The weights of ResNet were initialized using Stochastics Gradient Descent's with standard momentum parameters. ResNet-50 was better than ResNet-18 and reached accuracy between 73%-88%, sensitivity value between 64%-96%.

3.1.3 Dilation

Dilation is one of the operators in the area of mathematical morphology. The effect of this operator on binary or grayscale images is enlarging the boundaries of foreground pixels using a structuring element. Mathematically, the dilation of A by B, denoted as $A \oplus B$, is defined in terms of set operation:

$$A \oplus B = \{z|(\hat{B})_z \cap A \neq \varnothing\}, \tag{3.5}$$

where $\varnothing$ is the empty set and B is the structuring element, $\hat{B}$) is the reflection of set B, and $(B)_z$ is the translation of B by point $z = (z_1, z_2)$ (Wang e Chung, 2018).

In Wang e Chung (2018), the authors proposed a neural network framework able to detect gliomas in Magnetic Resonance (MR) brain images. Gliomas are primary brain tumors frequently found in adults, and the accurate identification of the affected regions is crucial in clinical diagnosis, treatment planning, and post-operation evaluation. Between the obstacles to accurate identification of gliomas, there is the variance of tumor size, shape, and location.

This approach used U-Net, a neural network that takes the full image context into account, and applied dilation to the ground truth of training samples. The dilation allowed the framework to learn the complex details of tumor structure in a coarse-to-fine approach. The U-Net architecture includes 3 convolutional layers of size 3x3, each of them followed by ReLU

activation function and batch normalization. Max-pooling layers and up-sampling of size 2x2 are adopted in the U-Net architecture too.

The authors observed that the dilation (1) expanded the tiny regions and connects the close but separated pieces, helping the network to focus on higher level features; (2) prevented overfitting because of the dynamic changes during the training; (3) the coarse-to-fine interface boosted the learning speed as well as the training efficiency. This approach showed improvement especially on rather small regions like tumor core and enhancing tumor and low computational cost for new test images.

The method was evaluated on the BRATS2015 dataset using the HG training set, which contains MR images from 220 patients. The HG images were randomly split into 132 images for training, 44 for validation, and 44 for testing. The dilation ratio used in the tests was $\alpha = 0.6$. The neural network model was implemented in Keras and Tensorflow backend and trained for 60 epochs using Adam optimizer, with a learning rate 8×10^{-5}. The model beat the state-of-the-art in Dice Coefficient, reaching an average detection accuracy of 78.38%.

In Ning et al. (2018), the authors proposed a Convolution Neural Network (CNN) containing a hierarchical dilation block for semantic image segmentation. Dilated convolution is a normal convolution that applies convolution filters with a whole, being an effective strategy to enlarge the size of receptive fields of the CNN and, consequently, making the CNN achieve better performances. Traditional mechanisms to apply dilated convolution are effective but with limitations, such as they are unable to capture the scale variations for objects in images.

Concerning to deal with this limitation in a effective way, the authors proposed a block, named HDblock, that contains multilevel parallel dilated convolutions and each convolution includes 3x3 convolution kernels with various dilation factors. One of the advantages of this method is the enlargement of the receptive field with less gains in the depth of deep neural networks. The proposed CNN was able to achieve real-time performance on images with full HD resolution.

The proposed model was constructed based on a lightweight architecture called darknet. The last three layers of the darknet were removed and the HDblock was appended. The authors employed Adam's optimization algorithm to train the network with 150 epochs, a learning rate set to 0.001, weight decay of 0.0002, and different batch size for each dataset: 4, 8, 12, and 16 for Kitti, CamVid, Helen, and Cityscapes, respectively. No data augmentation strategy was employed. The mean accuracy for each dataset was 71.5%, 71.0%, 87.2%, and 71.5% for Kitti, CamVid, Helen, and Cityscapes, respectively, and the running speed was 2x faster than ENet, the fastest neural network architecture designed for semantic segmentation until then.

In this chapter, we presented the main preprocessing algorithms used in our work. In the next chapter, we will present the image feature extractor methods applied in our work.

3.2 FEATURE EXTRACTORS

Feature extraction involves simplifying the number of resources required to describe a large set of data accurately. When performing analysis of complex data, one of the major problems stems from the number of variables involved. The need for computational power and amount of memory generally increases when the analysis is made with more variables and, indeed, may result in a classification algorithm with over fits of the training sample and generalizes poorly to new samples.

Feature extraction is a method of constructing combinations of the variables to deal with these problems and describes the data with sufficient accuracy. In this chapter, we explain the feature extractors used in our work. We start explaining the LBP (Local Binary Pattern) and its

two variants: ULBP (Uniform Local Binary Pattern) and RLBP (Robust Local Binary Pattern). The next explained feature extractor is GLCM (Gray Level Co-occurrence Matrix).

3.2.1 Local Binary Pattern

Local Binary Pattern (LBP) algorithm is a kind of local texture feature extraction method. It creates binary coding according to the grey value difference between the center pixel and neighborhood pixels in the sampling area and is widely used in image texture feature analysis. The original LBP operator is defined in a rectangular neighborhood with a size of 3×3, and the arbitrary color images should be converted into grey images with a grayscale value of [0,255] (Xu et al., 2012).

Pixels of the rectangular area are used as sampling points, denoted grey value of the center pixel as f_0, and grey-scale value of 8 pixels around it as $f_1, f_2, ..., f_8$. When $f_i \geq f_0$ the corresponding position is encoded as 1, and when $f_i < f_0$ the corresponding position is encoded as 0. After all pixels within the area are coded, the encoding value of 8 pixels around the center pixel will be composed of a binary number in a clockwise direction. The LBP coding can be used as features to reflect the texture information in images (Guo e Zhang, 2010). Figure 3.1 shows a schematic diagram of LBP algorithm.

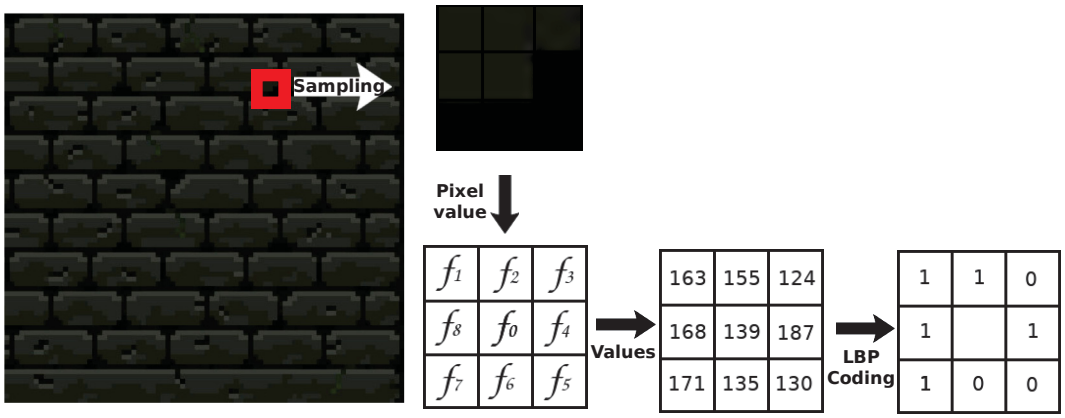


Figura 3.1: Diagram of LBP operator coding process

LBP has small computational complexity and shows great advantage when applied to texture retrieval in some cases.

Despite LBP reached impressive classification results working on a representative texture database, there are some issues uncovered by the algorithm. For example, LBP is noise sensitive and often characterizes different structural patterns with the same binary code, reducing the discriminability (Zhao et al., 2013b). Aiming to solve these flaws, new LBP-based feature extractors has been developed, between then Uniform Local Binary Pattern (ULBP) and Robust Local Binary Pattern (RLBP).

A **Uniform Local Binary Pattern (ULBP)** is based on the original LBP and can describe uniform patterns. Is one of the basic LBP extensions. An LBP is uniform when contains at most two 0-1 or 1-0 transition when viewed it as a circular bit string (Ojala et al., 2002). For example, the patterns 00000000 (0 transition), 11111110 (1 transition) and 11100111 (2 transitions) are uniform, whereas 00110010 (4 transitions), 10110110 (5 transitions) are not.

Robust Local Binary Pattern (RLBP) is an LBP-based feature descriptor that locates the possible bit in the LBP pattern changed by the noise and then revises the changed bit of the LBP pattern (Chen et al., 2013). Given an image I, let its histogram of LBP(8,1) be H, which has

256 bins before mapping it to a uniform pattern. For each bin H_i of $H(i = 0, ...255)$, we search all of its neighboring three-bit substrings, and map its y_3 or y_6 to y'_3 or $y'6$, respectively.

The mapping function can be denoted as follows:

$$RH_s = \Psi(H_i) \tag{3.6}$$

where RH is the resulted histogram of RLBP, composed of uniform patterns, and RH_s is the s-bin, $s \in \{1, ...58\}$. Let $|RH_s|$ be the occurrence frequency of RH_s. The occurrence frequency of RH_s is then changed to $|RH_s| + |H_i|$ after this mapping.

The mapping function $\Psi(H_i)$ is not a one-to-one mapping. Especially, one bin H_i would map to more than one bin (e.g., the LBP string 11010011 can be mapped to 11110011 or 11000011). Let T_i denote that the number of bins that H_i maps to those bins in RH by Eq. 3.6. Thus, Eq. 3.6 is revised as follows:

$$\{RH_{st}|t = 1, ..., T_i\} = \Psi(H_i), \tag{3.7}$$

and the new occurrence frequency of RH_{st} is computed as:

$$|RH_{st}| = |RH_{st}| + \frac{1}{T_i}|H_i|, t = 1, ..., T_i \tag{3.8}$$

Figure 3.2 shows the comparison between LBP and RLBP extractors when applied to a texture image. The noise treatment changes the pixel with a value of 124 to 139. With this change, the final binary string contains fewer bitwise when compared with LBP binary string.

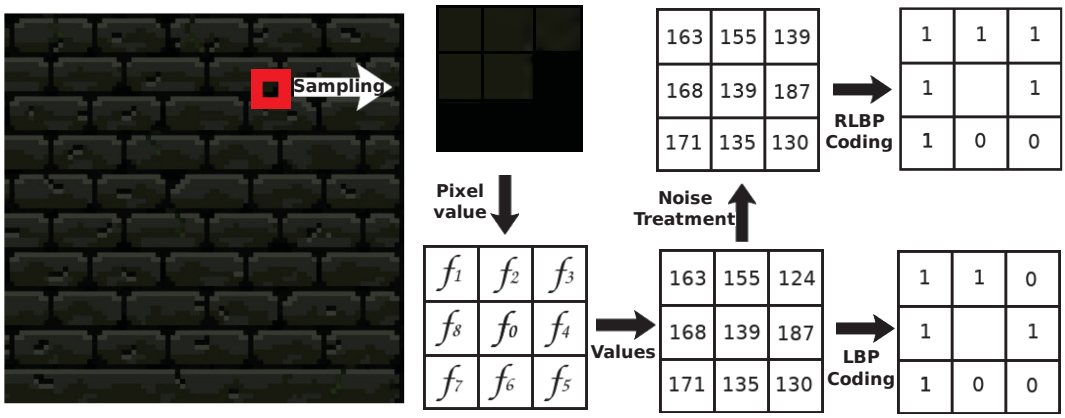


Figura 3.2: RLBP and LBP applied to a texture image

In Rasras et al. (2021), the authors created an LBP-based approach to obtain a signature to retrieve digital signals, such as digital color images and digital wave signals, commonly used in computer applications including computer security. Applying LBP as a feature extractor of the input digital data, the generated histogram can be used as an input data set to generate the digital file features, increasing the voice recognition ratio.

In Huang et al. (2020), the authors were concerned in improving Hyperspectral Image (HSI) classification, an important task in, for example, geological exploration, crop detection, and national defense. During the hyperspectral analysis, the Hughes phenomenon and the variability and the complexity of spectral features create a challenge in HSI classification. For the sake of explanation, the Hughes phenomenon occurs when the number of operational bands increases, causing an undesirable behavior on the classification accuracy: at a first moment the accuracy increases, following by a decrease.

To deal with this challenging scenario, the key to a successful HSI classification is the feature extractor process (Huang et al., 2020). Using LBP to directly extract texture features of HSI, the edge extraction capabilities of LBP help to improve the classification accuracy.

In Badaghei et al. (2021), the authors applied three feature extractions over traffic images aiming to recognize bickers without a helmet. One of them was ULBP, once a biker head image can be represented as a combination of micro-patterns by an LBP histogram. They divided the image into neighboring cells and calculated the LBP histogram in each cell, concatenating them all in a single histogram at the end. Using this feature extraction process, the classification accuracy was 98.03%.

In Srivastava et al. (2020), the authors proposed a Local Binary Pattern (LBP) based novel image hashing technique. After the preprocessing image step, the authors used ULBP to extract the features of the analyzed image. To do so, the image is divided into a cell size of 64x64 and the size of neighborhood pixels is 8.

3.2.2 Gray Level Co-occurrence Matrix

Known as one of the popular estimating methods to reveal the image properties (Liu et al., 2021), Gray Level Co-occurrence Matrix (GLCM) is a matrix where the number of rows and columns is equal to the number of gray levels, G, in the image. The matrix element $P(i, j|\Delta x, \Delta y)$ is the relative frequency with which two pixels, separated by a pixel distance $(\Delta x, \Delta y)$, occurs within a given neighborhood, one with intensity i and the other with intensity j. The matrix element $P(i, j|d, \theta)$ contains the second-order statistical probability values for changes between gay levels i and j at a particular angle (θ).

Using a large number of intensity levels G implies storing a lot of temporary data, i.e. a G x G matrix for each combination of $(\Delta x, \Delta y)$ or (d, θ). Due to their large scalability, the GLCM's are sensitive to the size of the texture samples on which they are estimated. Thus, the number of gray levels is often reduced. Equation 3.9 contains the math formula of GLCM, and the Figure 3.3 illustrates a calculation example of GLCM over a given image.

$$P(i, j|d, \theta) = \begin{Bmatrix} (x, y)|f(x, y) = i, f(x + \Delta x, y + \Delta y = j); \\ x, y = 0, 1, 2, ..., N - 1 \end{Bmatrix} \tag{3.9}$$

In Aouat et al. (2021), the authors developed an approach for texture segmentation based on GLCM. After applied geometric transformations on the image (change of scale and rotation), the next step of the proposed model is the application of GLCM. Covering the texture image with an analysis window containing a size **T** and offset **d**, the goal is to calculate the GLCM of each analysis window and calculate the Haralick features, assigning them to the central pixel of the analysis window.

The Haralick features used in their approach were energy, homogeneity, correlation, and contrast, and the size **T** was empirically obtained, testing the following scales: 5x5, 9x9, 15x15, and 20x20. With bigger window sizes, the textures were not well recognized because one window contained more than one texture. The size **T** was set to **15x15**, once this window size gave better and homogeneous results. The offset **d** also was empirically obtained, testing values between 1 and 5. The more the offset was increased, the more the quality image loss, affecting the segmentation accuracy. The offset 1, 2, and 3 gave good results, so **d** was set to **3** to reduce the computation time.

After that, the authors used a gradient for edge detection from the four extracted images (one for each Haralick feature.) Following, two images were calculated (vertical and horizontal direction) and generated the mean of these two images, combining them in one output image. A

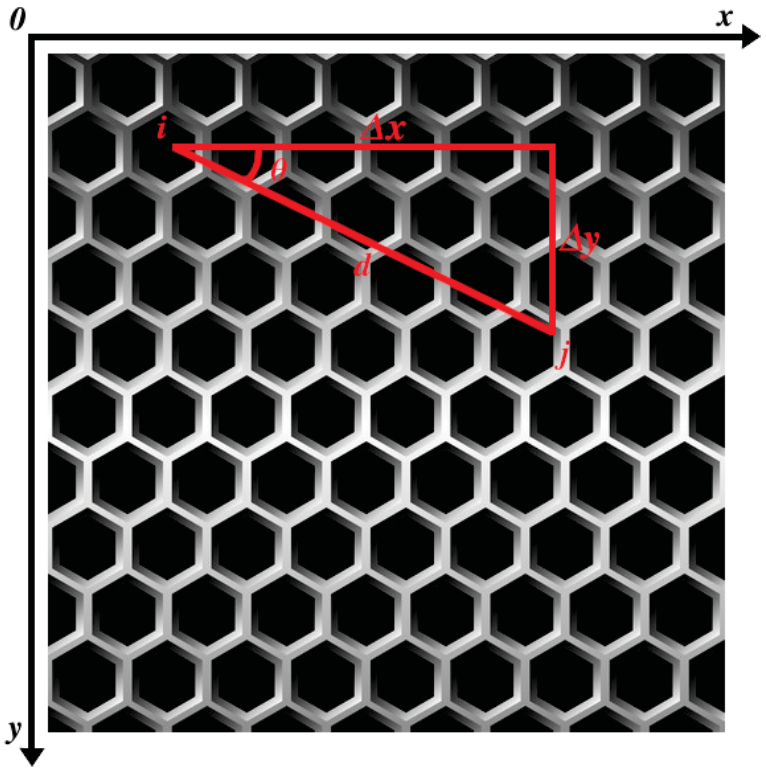


Figura 3.3: Calculation example of GLCM

threshold was applied on this output image, turning all the pixels with a value greater than **S** into 255, or 0 otherwise. The **S** value was empirically set to **50**.

To improve the segmentation method, smoothing filters were applied to remove noise, and the hysteresis thresholding to keep the strongest edges of the image and keep their continuity. A morphological closing process was applied to keep the edges close, derived from dilation and erosion. To solve the discontinuity problem between the edge pixels, the authors joined the nearest points according to a distance **d** using a circle analysis form with radius **r**. Finally, to improve the edge image and remove the false detected regions, the authors proposed a growing region method to localize textures, using a circle with radius **R** to delineate areas of the image. The proposed approach obtained 0.97 and 0.92 of overlap ratio average for the Brodatz dataset and the database presented by Oulu University, respectively.

In Hamouchene et al. (2014) was proposed a decomposing architecture for the texture matching task. The decomposing architecture, as generally used in the literature, consists of fractionating the image into fixed-size blocks, like 16x16 or 32x32. The authors, otherwise, created an approach that used dynamic decomposition size combined, between other, with GLCM.

The GLCM method was used to extract the features from the decomposed regions. The proposed approach used the circular geometric shape to define the decomposing blocks once it resulted in a better segmentation compared with square shapes. The authors generated test images, named by the authors as query images, using the Brodatz texture images dataset, randomly selecting textures on this dataset. The query texture images were well recognized and good segmentation results were obtained with the proposed approach.

3.2.3 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised statistical technique introduced by F.R.S. (1901) as a tool for simplifying the scalability of the data while maintaining its variance. At the same time, PCA can extract relevant patterns in the data while retaining interpretability and minimizing information loss (Tibrewala et al., 2020). PCA extracts the fundamental properties of a linear system by the single value decomposition method. This technique has been applied to discard noise data in digital signal processing, image recognition, and classification problems (Ng, 2017).

The covariance matrix is first created using the training images. Next, the eigenvectors and corresponding eigenvalues are computed. Finally, the test images are identified by projecting these into the subspace and comparing them with the trained images in the subspace domain.

The PCA algorithm follows the steps:

- Let an input image $X(x, y)$ be a two scale m x n array of intensity values. The average image of the training set is defined by

$$\bar{x} = \frac{1}{L} \sum_{i=1}^{L} x_i. \tag{3.10}$$

- Calculate the Covariance matrix to represent the scatter degree of all feature vectors related to the average vector. The Covariance matrix C is defined by

$$C_{xx} = \frac{1}{L} \sum_{i=1}^{L} (x_i - \bar{x})(x_i - \bar{x})^T. \tag{3.11}$$

- The Eigenvectors and corresponding Eigenvalues are computed by

$$CV = \lambda V, \tag{3.12}$$

  where V is the set of Eigenvectors associated with its Eigenvalue λ.

- Sort the Eigenvector according to their corresponding Eigenvalues from high to low.

In Kamencay et al. (2016), the authors developed a system to recognize animal images. Three feature extractors were tested, one of them being the PCA. When the recognition system used PCA, the covariance matrix is first created using the training images. Next, the eigenvectors and corresponding eigenvalues are computed. Finally, the test images are identified by projecting these into the subspace and comparing them with the trained images in the subspace domain.

To validate the methodology, the authors used a created animal dataset containing images of 5 different classes of different animals: bear, hog, deer, fox, and wolf. Each class had 60 different images, totaling 300 images, each of them with size of 130x130 pixels. The authors executed experiments varying the proportion of images used as training images and testing images. The PCA recognition rate increased when the number of training images was increased too, reaching 82% of accuracy when 98% of the dataset was used for training.

In the next section, we describe the types of ML algorithms, explain how the ML classifiers used in SCTRx works and explain the steps to apply ML algorithms in computer networking.

3.3 MACHINE LEARNING

Computer networking often deals with complex problems that demand efficient solutions. So bring ML algorithms to the network domain is a promising idea to achieve higher performance in network execution. Between the main network problems where ML for networking are suitable and efficient, we can found the performance prediction. Network scheduling and parameter adaption can also be performed by ML throw decision-making algorithms (Wang et al., 2018).

ML can be successfully applied to four broad categories of problems: *clustering*, *rule extraction*, *classification* and *regression* (Boutaba et al., 2018). The main goal of a clustering problem is to group similar data while increasing the gap between the groups. Rule extraction problems concerns identifying statistical relationships in data. Regression problems aim to map a set of new input data to a set of continuous-valued output. Classification problems have the same concern of mapping a set of new input data but to a set of discrete output.

In ML and pattern recognition applications, the processing of high scale data requires large computation time and capacity storage. Though, it leads to poor performances when the scalability to sample size ratio is high. To improve performances, the sample scalability is reduced thanks to feature extraction schemes. Feature extraction transforms the original input into a new low scalabe space by combining the initial features, while feature selection retains the most relevant ones to build a low scalabe feature space. ML algorithms belong to one of the following learning paradigms, each of them composed by a distinct feature extraction scheme: *supervised learning*, *unsupervised learning*, *semi-supervised learning* and *reinforcement learning* (Wang et al., 2018; Boutaba et al., 2018; Kalakech et al., 2011).

Supervised learning uses labeled data for knowledge construction, being useful to classification and regression problems. Using labeled information to train the algorithms, supervised learning methods generally achieve better results than unsupervised learning (Sheikhpour et al., 2017). However, acquiring such labeled information is hard to obtain (there are a lot of unlabeled data in the real world, but it is necessary for expertise people to label them) and is an expensive process because a lot of labeled data is necessary.

In Liu et al. (2007), the authors used an unsupervised K-means clustering algorithm to classify the network traffic. K-means algorithm uses statistical information as an input vector to construct the classification models, starting with a training set and an assigned number of clusters. A metric, i.e. distance, is considered to evaluate the similarity between the training examples. Liu et al. chose the Euclidean distance as a similarity measurement. The authors used different datasets (all of them composed of TCP-based applications). The classes considered in the experiment are WWW, MAIL, P2P, FTP-CONTROL, FTP-PASV, ATTACK, DATABASE, FTP-DATA, SERVICES, INTERACTIVE, MULTIMEDIA, and GAMES. The overall accuracy reached 80% and before a log transformation, 90% or more.

Roughan et al. worked with traffic classification too but using a supervised strategy (Roughan et al., 2004). It is one of the first works involving traffic classification that used the ML approach. They employed k-Nearest Neighbors (k-NN) and Linear Discriminant Analysis (LDA) to recognize different classes of packets based on QoS requirements. The classes addressed in the experiment are Telnet, FTP-data, Kazaa, RealMedia, Streaming, DNS, and HTTPS. The error rate was 5.1 and 9.4% using the classifiers k-NN and LDA, respectively.

Eerman et al. propose a semi-supervised TCP traffic classification technique (Erman et al., 2007). Their approach designed classifiers trained with few labeled data and many unlabeled flows. The dataset used in this work consists of empirical Internet traffic traces that span 6 months. With this approach, a variety of applications including Web, P2P file sharing, email, and FTP could be correctly classified with accuracy over 98%.

3.3.1 Random Forest

The Random Forest algorithm is an ensemble classifier dated from 2001 when Leo Breiman formalized it In Breiman (2001). Being a successful algorithm as a general-purpose classification and regression method and with few parameters to tune, Random Forest reached popularity in a short time. In problems where the number of variables is much larger than the number of observations, Random Forest has shown excellent performance. Chemoinformatic, ecology, 3D object recognition and bioinformatics are examples of areas that Random Forest was successfully applied (Scornet, 2017).

Random Forest is a supervised learning procedure that operates according to the "divide and conquer"principle. In the first step, the algorithm generates a random vector sampled independently of the input vector. For each fraction, a randomized tree predictor is grown using one of the random vectors as a base. So, each tree casts a unit vote for the most popular class to classify an input vector. In the end, Random Forest aggregates these predictors together, using the average to generate the final result.

To generate every single tree, Random Forest algorithm works as follows: if the number of records in the training set is N, then N records are sampled at random but with replacement, from the original data, this is a bootstrap sample. This sample will be used to train the growing tree. Being M the number of input variables, a number $m \ll M$ (in literature, m is often referenced as *ntry* or k) is selected such that each node, m variables are selected at random out of M and the best split of these m attributes will be used to split the node. The value m will be held constant during the forest growing. The trees will be generated as large as possible, with no pruning. The number of trees is another pre-defined variable, generally named as *Ntree*. The last parameter used by Random Forest is *nodesize*, which controls the depth of the tree (i.e. number of instances in the leaf node) and is usually set to one (Kulkarni e Sinha, 2013).

Farnaaz et al. built an Intrusion Detection System (IDS) using Random Forest. The main goal of IDS is to find intrusions and classify the type of attack on computational systems. They used the NSL-KDD dataset (a dataset consisting of legit and malicious network packets with 42 attributes, where the last attribute is the class label) in ARFF format to validate the proposed approach. Setting $Ntree = 100$, Random Forest reached 99.67% of accuracy detecting DoS, Probe, R2L and U2R attacks (Farnaaz e Jabbar, 2016).

Yu et al. developed a Random Forest approach for predicting air quality (RAQ) in urban sensing systems. The data used for their experiments is a public dataset generated from Shenyang's urban sensing systems, with a one-month range of collection (4 May 2015 to 5 June 2015). With $Ntree = 400$ and $ntry = 4$, the overall precision of Random Forest for Air Quality Index (AQI) prediction is 81% (Yu et al., 2016).

Osin et al. proposed a solution to the problem of predicting the quality of a computer network using Random Forest. The dataset used in their experiments was captured in November 2014 from network traffic of two company servers: a web application server and a content server. Random Forest result indicated the prediction of a decrease in the quality of the IP network: 0 indicates that the network operates normally and 1 indicates loss of packets and a repeated request for the same connection. Random Forest reached 70% of accuracy in detection network troubles (Osin e Sheluhin, 2019).

3.3.2 Support Vector Machine

Introduced in 1995, when Vladmir Vapnik and Corinna Cortes published the paper (Cortes e Vapnik, 1995), Support Vector Machine (SVM) is one of the state-of-the-art ML techniques. SVM algorithm exhibited great prediction accuracy and prominent modeling

capability in a wide range of real classification and pattern recognition (Faris et al., 2017), such as document classification (D'Orazio et al., 2014), image classification, outlier detection and sleep monitoring system (Alickovic e Subasi, 2018).

SVM is a supervised ML algorithm that can be applied to linearly and non-linearly separable patterns. The main idea behind SVM is the construction of an optimal hyperplane, which can be used for classification, for linearly separable patterns. For non-linear separable problems, the given pattern by transforming it into a new space usually a higher scale space so that in higher scale space, the pattern becomes linearly separable. This transformation can be achieved by using various nonlinear mapping such as polynomial, sigmoidal, and Gaussian (Pradhan, 2012), (Kecman e Wang, 2005). SVMs do not have predefined parameters and their number depends on the training data used (Kecman e Wang, 2005). Both in linear and nonlinear patterns, SVM aims to find the maximal margin of separation between the classes. For demonstration, in Figure 3.4 there is a representation of the hyperplane and the maximal margin for a classification problem involving two classes.

Ye et al. used SVM to detect DDoS attacks in SDN. In their experiments, OpenFlow switches and a Floodlight controller were used to create an SDN topology in a simulated environment (made with Mininet). In their simulation, legit packets traffic over the topology concurrently with DDoS packets. Both types of traffic used TCP, UDP, and ICMP packets. With SVM, they reached an average detection of 95.64% and 1.26% of average false alarm rate (Ye et al., 2018).

In Yang (2018), Yang proposed a novel anomaly network traffic detection algorithm under the cloud computing environment based on an SVM classifier. To develop this approach, he considered six features: number of sources IP address; the number of source port number; the number of destination IP address; the number of destination port number; the number of packet type; and number of network packets. Combining SVM with hybrid information entropy, the approach named Ent-SVM reached high accuracy using both KDDcup99 and NSL-KDD datasets.

Zhang et al. analyzed the phenomenon of network under Low-rate denial-of-service (LDoS) attack, a new kind of network attack with a low average with low average attack traffic and high concealment, and proposed a new SVM-based framework. Using Principal Component Analysis (PCA) to extract the main features, the SVM classifier was trained with the principal components of the original flow data. The authors conducted multiple experiments on NS2, simulating a topology with three routers, and reaching an average accuracy of 96.68%. In a test-bed environment, with two routers, one server, and four hosts (one of them being an attacker), the average correct detection rate was 95.37% (Zhang et al., 2019).

In this section we showed the preprocessing algorithms, feature extractors, and ML approaches used to implement our proposal. In the next chapter, we explain our proposed framework (SCTRx) to deal with the limitation of TReco, the evaluations made with SCTRx, a discussion about the results and future works.

3.4 TRECO

The Traffic Matrix Recognition (TReco) framework was designed to identify which HPC application is using the network and to reprogram the switches for optimizing the network according to the application's communication pattern.

TReco has two operation modes, the first is used for "`learning`" the communication patterns and the second is the "`running`" mode. The first stage of a pattern recognition system

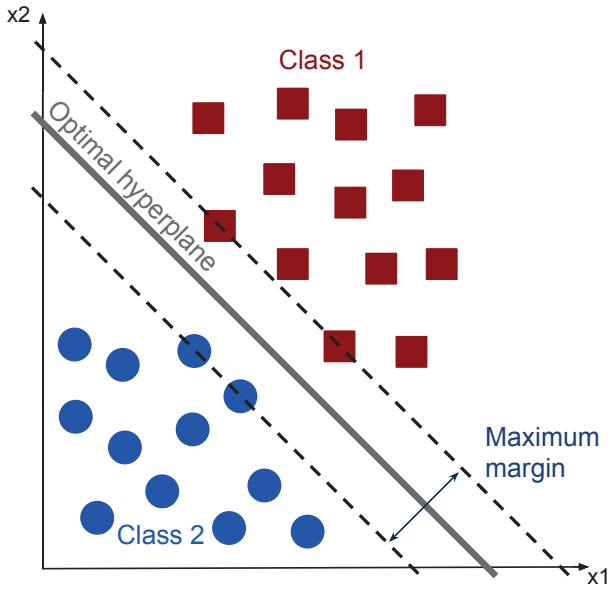


Figura 3.4: Maximal margin and hyperplane in SVM

is data acquisition. The input depends on collecting the applications' TMs. If the TM are collected every time t, the application execution can be seen as a set of multiple TM $M_{[i][j]}(t)$.

TRECO supports two methods for collecting the TM, by (a) installing traffic measurement rules in each SDN-enabled switch, and at every time t, the switches' flow tables were dumped to files, and then, these files were parsed for generating the TM. (b) Reading dump files from the disk. An important remark is that the manner TM are collected is orthogonal to TRECO, which means that it might be obtained using different techniques (Gong et al., 2015; Trois et al., 2016).

Next, two processes occur in parallel, (i) all acquired TM are displayed to the user that identifies the communication patterns, grouping the TM in those patterns, as shown in Figure 3.5. (ii) For preprocessing the TM, TRECO used the unscramble method (Trois et al., 2018), which generates the textures for each communication pattern, then, two visual descriptors, ULBP (Ojala et al., 2002) and RLBP (Zhao et al., 2013a) were applied for extracting the feature vectors used as input for the ML classifiers.

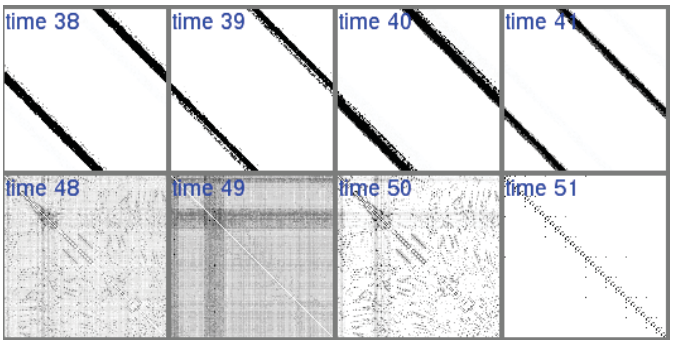


Figura 3.5: Selecting and grouping the TM patterns.

TRECO uses the Unscrambling Function as preprocessing algorithm. Unscrambling Function reorders the TM lines and columns, rendering a different texture for each communication pattern. The basic premise of this function is, for each line, to "bring" the most communicating pair of nodes close to the matrix main diagonal. As our assumption is to classify the applications based on already known TM, so we use the supervised classifiers SVM and RF.

Learning new communication patterns requires a minimum number of TM which may vary with the patterns already learned. So, to be sure that the amount of TM acquired is sufficient, and that TRECO will be able to classify the new pattern in the running mode, an accuracy performance verification can be executed. In this stage, the new communication patterns are used, along with the previously learned patterns, for feeding the classifier, returning the prediction accuracy. If the user considers the results adequate, the classifier training files are updated with the newly learned patterns.

The second operating mode of TRECO is the running mode; in this case, the system continuously acquires the TM on the fly, applies preprocessing function, extracts the feature vectors, and classifies the communication pattern (Figure 3.6).

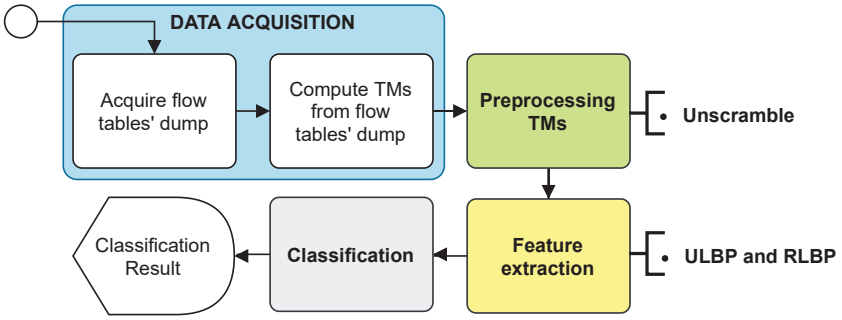


Figura 3.6: TRECO's flowchart for classifying the applications.

For an in-depth understanding of how communication patterns could be represented by TM, in the work Trois (2017); Trois et al. (2018) the authors investigated a set of HPC applications (MapReduce and scientific applications) and some benchmarks developed by NASA Advanced Supercomputing Division, called NPB[1] (Bailey et al., 1991), analyzing their TMs and execution phases. However, this work did not consider that the applications could be executed on different scales, meaning that the size of the TMs used for training was the same as the test ones.

We tackle a future direction reported in Trois (2017), wish required the ML classifier to be trained with TMs of different scales for each class. Therefore, our work studies methods for classifying the applications with a variable number of nodes, which is fundamental for its use in real scenarios.

[1]The NAS Parallel Benchmarks are set of applications designed to help evaluate the performance of parallel supercomputers, consisting of five kernels and three pseudo-applications. Available at: `https://www.nas.nasa.gov/publications/npb.html`

4 SCTRECO

SCTRECO [1] is an evolution of the TRECO framework. SCTRECO' operation, showed in Figure 4.1 contains similarities with TRECO, but focus to deal with different TMs scales. SCTRECO's data acquisition occurs like in TRECO. After that, SCTRECO can be divided in three steps: preprocessing the TMs; feature extraction; and classification.

To deal with TMs of multiple scales, SCTRECO explores more preprocessing algorithms than TRECO, but maintains the same feature extractors (ULBP and RLBP) and ML algorithms (SVM and RF) used in TRECO. We tried two additional feature extractors: GLCM and PCA. As described in the Section 3.2, these two feature extractors achieve good recognition rates when applied to texture classification problems.

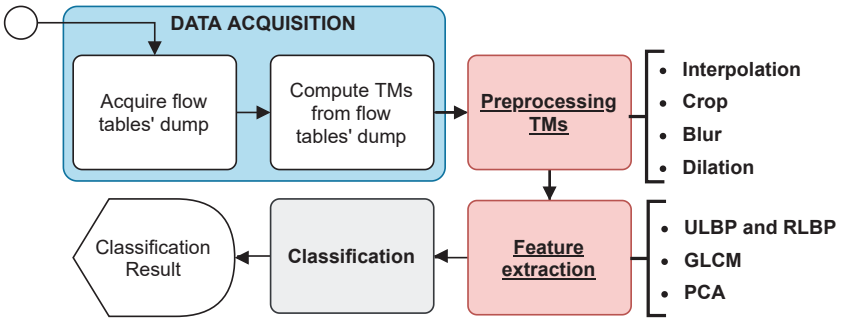


Figura 4.1: SCTRECO's flowchart for classifying the HPC applications.

It is important to say that our framework works in HPCs where each processes are executed in individual nodes. As the basis of our solution is the capture and analysis of the packages trafficked between the SDN switches, our framework would't be able to recognize the applications if there are processes running on the same node.

Despite the addition of the two feature extractors, we concentrated efforts on trying to find a preprocessing strategy that would allow the recognition of matrices in a scale-free scenario. Since TRECO already combines feature extractors and classifiers that have good TMs recognition rates, we chose to try to make the TMs as similar as possible before feature extraction and classification.

This chapter contains six sections. In Section 4.1, we explain how the dataset was obtained and how we made our evaluations. In Section 4.2, we describe how SCTRECO works using interpolation as preprocessing algorithm, showing the evaluations when the TMs are interpolated. Section 4.3 shows how SCTRECO blurs the TMs and contains the evaluation of this method. In Section 4.4, we evaluate how SCTRECO works cropping the test TMs. Section 4.5 describes the dilation process used as preprocessing algorithm and evaluate this. Finally, Section 4.6 compares SCTRECO and TRECO results and shows the improvements.

4.1 DATASET ACQUISITION AND EXPERIMENTS DEFINITION

To evaluate our framework, we used a dataset obtained by ourselves. We selected the TMs of three of the most network-intensive scientific applications implemented with the OpenLB library Heuveline e Latt (2007): a) **Multicomponent** (*multc*) demonstrates the instability

[1] SCTRECO is available at: `https://github.com/rafaelGomesCastro/trex`

generated by a heavy fluid penetrating a light one; b) **Cylinder** (*cyl2d*) simulates flow passing through a circular cylinder; and c) **Bifurcation** (*bifu*) reproduces human lungs bifurcations. We performing tests with a different number of processes (128, 256, and 512), running them for a period of 30 minutes, collecting their TMs every second.

The collect process resulted in a dataset with three classes, each of them with TMs with scales 128x128, 256x256 and 512x512. Finishing the dataset collection, we manually select the most significant TMs, which means the TMs with more communications. At the end, the dataset contains 1026 TMs of *bifu* application (392 of them with scale 128x128; 322 with 256x256; and 312 with 512x512); 1009 TMs of *cyl2d* application (263 with 128x128; 468 with 256x256; and 278 with 512x512); and 3043 TMs of *multc* application (1431 with 128x128 and 1612 with 256x256). Aiming to train our framework with a balanced dataset, in each evaluation we randomly selected 263 TMs for each class and used accuracy as metric.

As shown in Figure 4.2, we evaluated each of the preprocessing algorithm individually. Only the dilation depends on the result of the interpolation, since, after executing the dilation, we resize the matrices to the scale that resulted in greater accuracy. Our evaluations started using 128x128 TMs to train the classifiers, while 256x256 TMs was used to test them. The goal of this first evaluation was identify the best configuration parameters for each preprocessing algorithm.

For the interpolation algorithm, we made evaluations changing the rescaling scale; for blur and dilation algorithms, we changed the kernel size. The idea behind using cropped is to use an algorithm that does not cause distortions in the resulting TMs. So, the crop algorithm didn't need parameter evaluations, once in all the evaluations we cropped a slice of the test TMs that contains the same scale of training TMs.

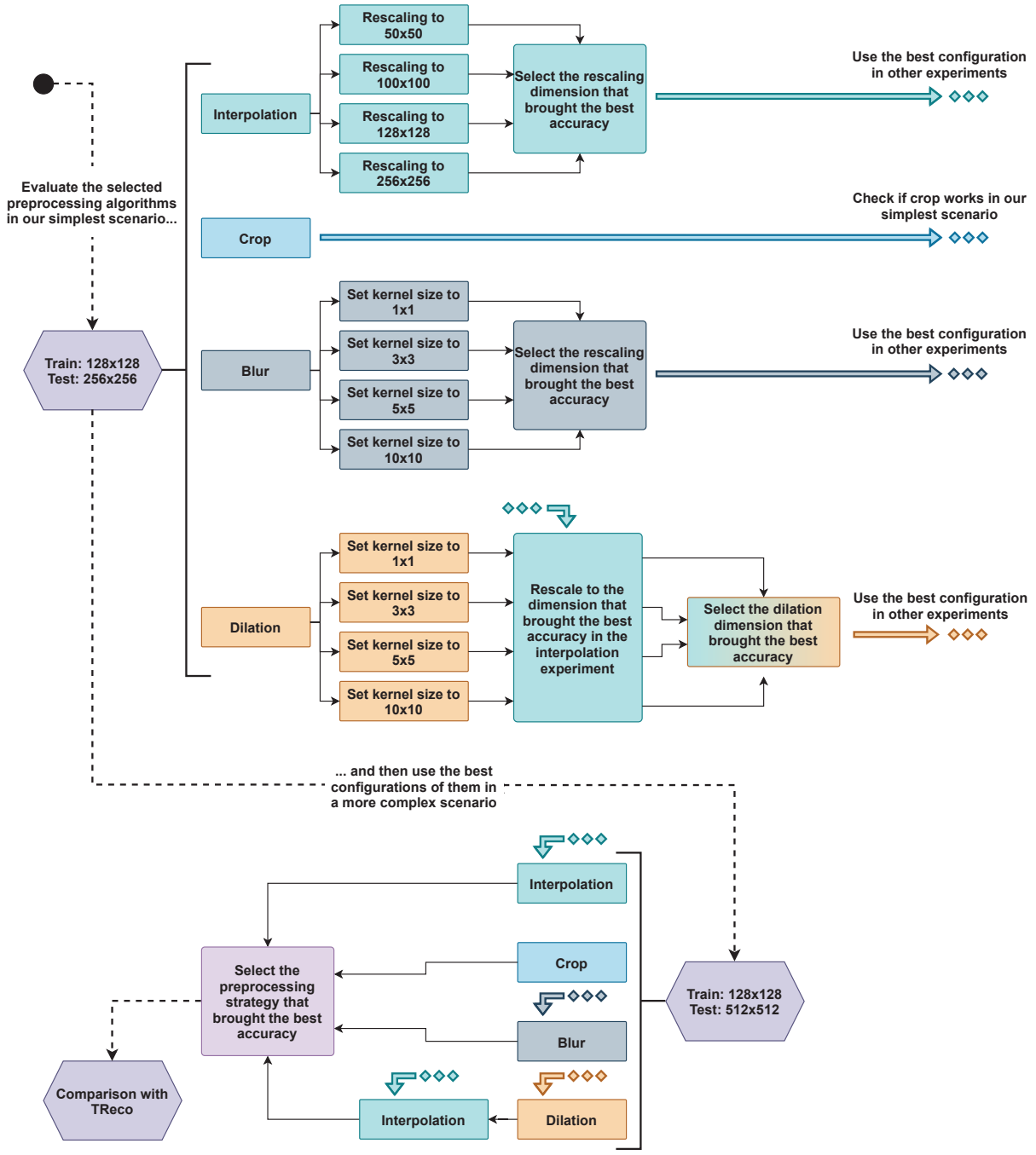


Figura 4.2: Evaluations' flowchart

After that, we evaluated SCTReco using 128x128 TMs to train the classifiers and 512x512 TMs to test them. Each preprocessing algorithm was set with the configuration parameters obtained in the previous evaluation. The purpose of this evaluation is analyse if our framework works fine in a scenario where the difference between the training and test TMs scales is bigger.

We finish our evaluations selecting the best SCTReco'results and comparing them with TReco's results. We ran TReco in the same environments, and added the feature extractors Eerman, Fahad and Soysal. In all of our evaluations, after grid search with cross-validation process, we set the SVM parameters *gamma* to 8.0 and *c* to 8192.0, and kernel was defined as RFB. For RF, we defined the 100 number of trees equal to 100 and Gini criterion to measure the quality of the splits.

4.2 INTERPOLATION

As TReco works well recognizing same scale TMs, our first preprocessing approach to brings more similarity between the training and test TMs was rescaling them to the same scale, as showed in Figure 4.2. We chose the nearest neighbor interpolation method to do the rescaling, described in Section 3.1, once this method is simple and cheap computationally when compared with other classical interpolation methods.

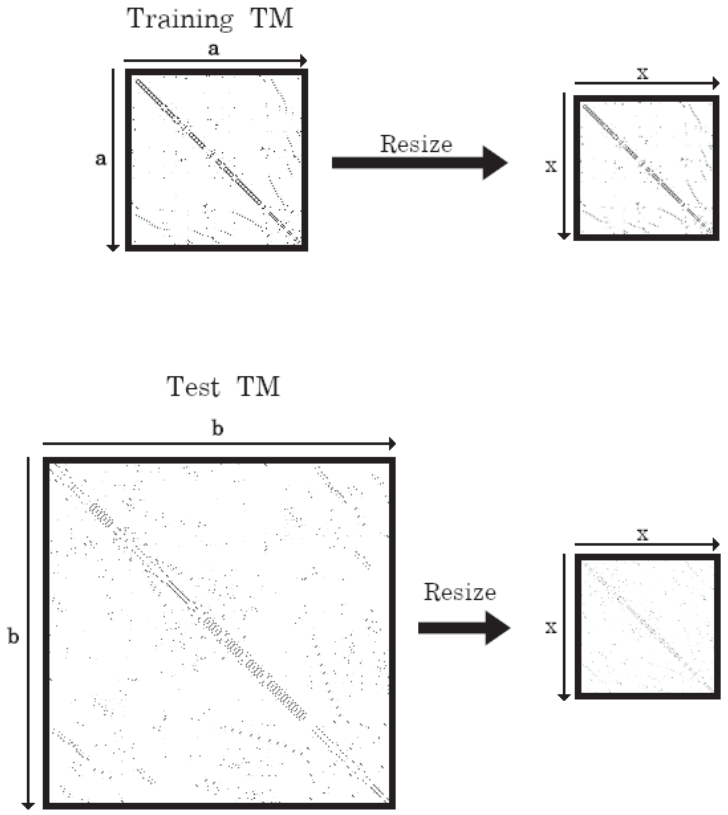


Figura 4.3: SCTReco's preprocessing step using interpolation.

Our preliminary evaluation of this preprocessing method, aiming to search the ideal resize scale, considered the classification accuracy value and a real dataset containing three classes of TMs (bifu, multc, and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 256x256, used to test. We rescaled the TMs for four scales: 50x50; 100x100; 128x128; and 256x256, and used ULBP, RLBP, PCA, and GLCM as feature extractors. The results, displayed in Table 4.1, containing the average results of ten executions, shows that rescaling the TMs to 100x100 reached the best accuracy value: 55.85%.

Rescaling	Classifier	ULBP (%)	RLBP (%)	PCA (%)	GLCM (%)
50x50	SVM	33.59	33.75	33.80	12.55
	RF	44.53	44.65	25.56	5.16
100x100	SVM	33.63	33.97	33.88	12.04
	RF	**55.85**	44.70	24.38	3.38
128x128	SVM	33.72	34.02	33.88	9.13
	RF	50.18	43.93	29.18	9.63
256x256	SVM	33.76	34.05	33.92	1.65
	RF	33.88	33.67	21.93	12.59

Tabela 4.1: Results of interpolation method rescaling the TMs for 50x50, 100x100, 128x128, and 256x256 scale.

Next, we evaluated the interpolation method using a real dataset containing two classes of TMs (bifu and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 512x512, used to test. We rescaled the TMs for scale 100x100, and used ULBP, RLBP, PCA, and GLCM as feature extractors. The results, showed in Figure 4.4, containing the average results of ten executions, show that PCA combined with SVM reached the best accuracy value: 50.44%. Compared with the previous evaluation, we can see that, except for ULBP and RF, the accuracy value increased. This behaviour occurred because the first evaluation contained three classes, while the second evaluation contained only two.

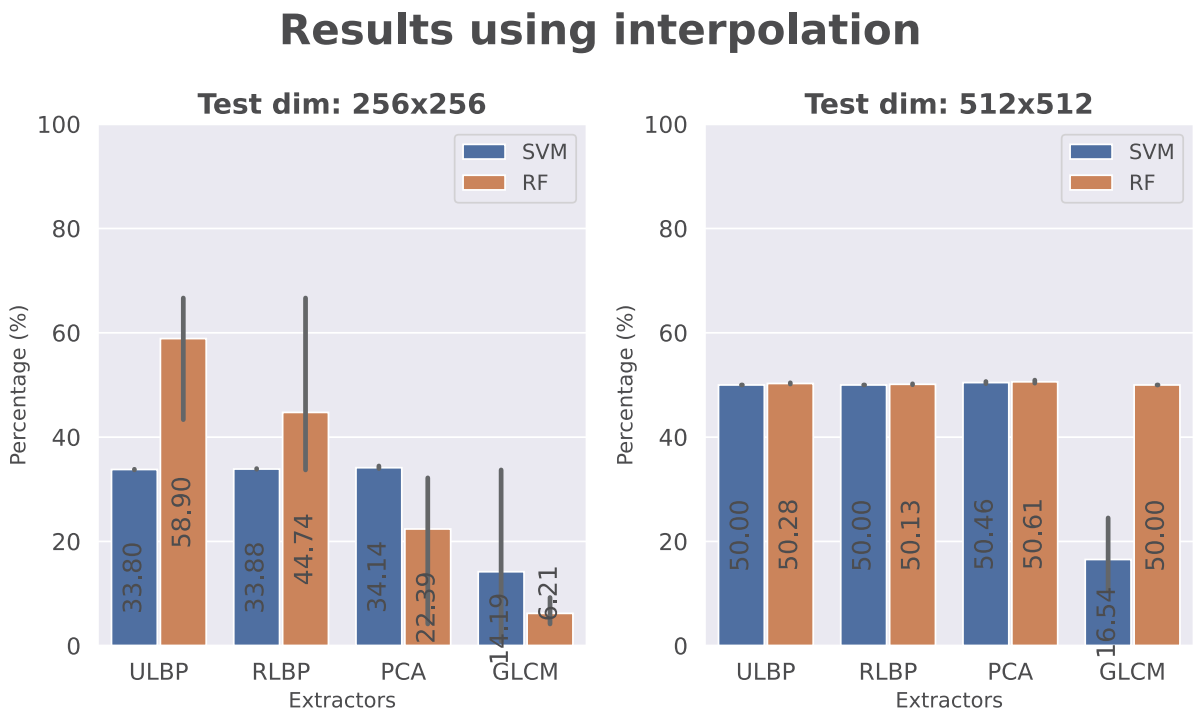


Figura 4.4: Classification results using 128x128 TMs for training and rescaling the TMs to 50x50. **1)** The left barchart shows the accuracy values classifying 256x256 TMs. **2)** The right barchart shows the accuracy values classifying 512x512 TMs.

4.3 GAUSSIAN BLUR

Our next approach used the Gaussian blur, described in Section 3.1, as preprocessing algorithm. We applied Gaussian blur to the TMs because this algorithm removes "outlier"pixels

that may be noise in the image, while leaving the majority of the image intact. As showed in Figure 4.5, we blurred the training and test TMs with the idea of disguising the differences and highlighting the similarities.

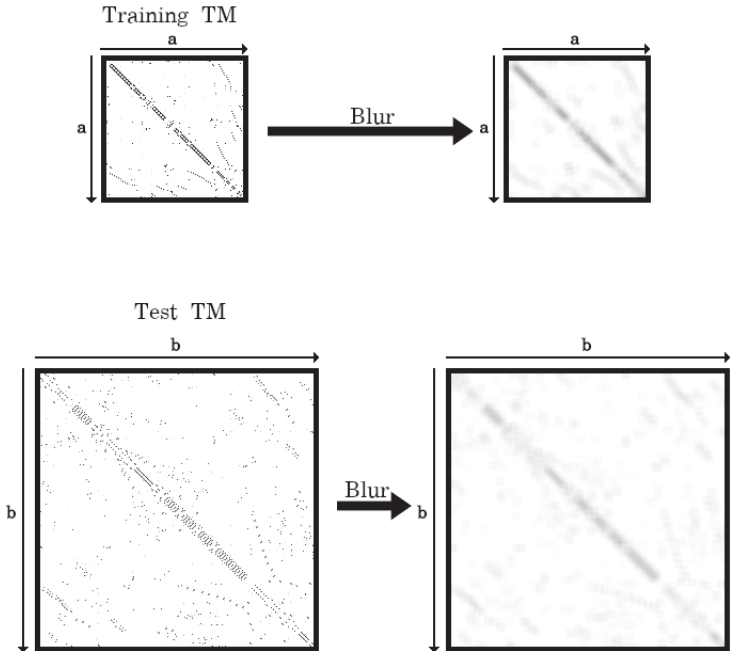


Figura 4.5: SCTReco's preprocessing step using Gaussian blur.

In the blur evaluation, we considered the classification accuracy value and a real dataset containing three classes of TMs (bifu, multc, and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 256x256, used to test. The kernel size evaluated was: 1x1, 3x3, 5x5, and 10x10; using ULBP, RLBP, PCA, and GLCM as feature extractors. As showed in Table 4.2, the best accuracy value, 78.88% was reached using blur with kernel size 3x3, SVM classifier and RLBP feature extractor.

Kernel size	Classifier	ULBP (%)	RLBP (%)	PCA (%)	GLCM (%)
1x1	SVM	33.95	33.77	2.48	33.59
	RF	47.13	39.00	21.09	33.33
3x3	SVM	65.61	39.67	2.83	44.70
	RF	51.54	33.88	14.58	42.97
5x5	SVM	65.91	**78.88**	1.52	33.50
	RF	54.84	65.44	6.21	33.33
10x10	SVM	33.92	33.55	2.28	33.46
	RF	38.74	39.29	18.25	33.33

Tabela 4.2: Results of blur method using kernel sizes 1x1, 3x3, 5x5, and 10x10.

Next, we evaluated the blur using a real dataset containing two classes of TMs (bifu and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 512x512, used to test. We applied blur with kernel size 3x3, and used ULBP, RLBP, PCA, and GLCM as feature extractors. The results, showed in Figure 4.6, containing the average results of ten executions, show that PCA and RF reached the highest accuracy value, 55.99%, but the other combinations reached close accuracy values. Using SVM as classifier, ULBP, RLBP, PCA, and GLCM reached

50.00%; using SVM as classifier, ULBP reached 50.17%; RLBP, 50.23%; PCA, 55.99%, and GLCM, 50.00%.

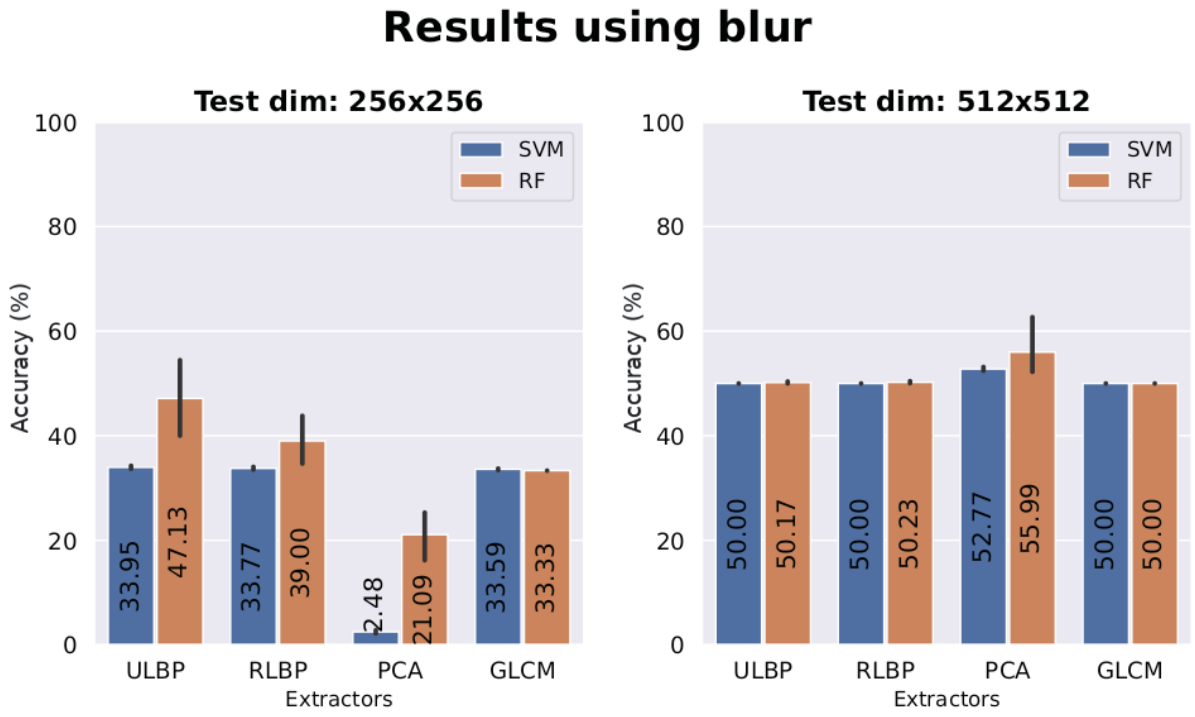


Figura 4.6: Classification results using blur preprocessing and 128x128 TMs for training. **1)** The left barchart shows the accuracy values classifying 256x256 TMs; and **2)** the right barchart shows the accuracy values classifying 512x512 TMs.

4.4 CROP

The next preprocessing algorithm evaluated was crop. As showed in Figure 4.7, we cropped just the test TMs, making them as the same scale as the training TMs. We applied crop aiming to evaluate a preprocessing algorithm that scale down the test TMs without cause distortions in the resulted TM. We noticed that the patterns present in the TMs that make up our dataset tend to be at the center of the TMs, so we chose to crop the center of the test TMs aiming to take the most relevant part of them. In this evaluation, the training TMs proceed unchanged for the feature extraction step.

To evaluate the crop algorithm, we considered the classification accuracy value and a real dataset containing three classes of TMs (bifu, multc, and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 256x256, used to test. The cropped area of the test TMs was the 128x128 middle pixels.

We made a second evaluation using the same crop algorithm, feature extractors and classifiers but using a different dataset. This dataset contains two classes of TMs (bifu and cyl2d) with scales 128x128, used to train the classifiers, and 512x512, used in test phase. As showed in Figure 4.8, using SVM to classify the 256x256 TMs reached 66.71% of accuracy with ULBP and RLBP, 55.64% with PCA and 13.45% with GLCM. Using RF, 61.01%, 50.26%, 75.42%, and 10.30% with ULBP, RLBP, PCA, and GLCM, respectively.

Classifying 512x512 TMs, SVM reached an accuracy value of 50.34% with ULBP as feature extractor, 50.00% with RLBP, 95.69% with PCA, and 16.35% with GLCM. RF reached 50.61% with ULBP, 50.25% with RLBP, 78.04% with PCA, and 12.07% with GLCM.

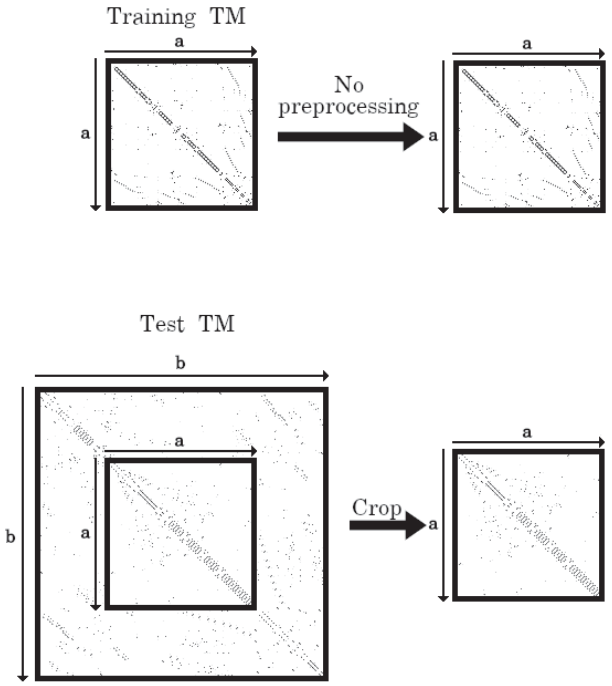


Figura 4.7: SCTRECO's preprocessing step using crop.

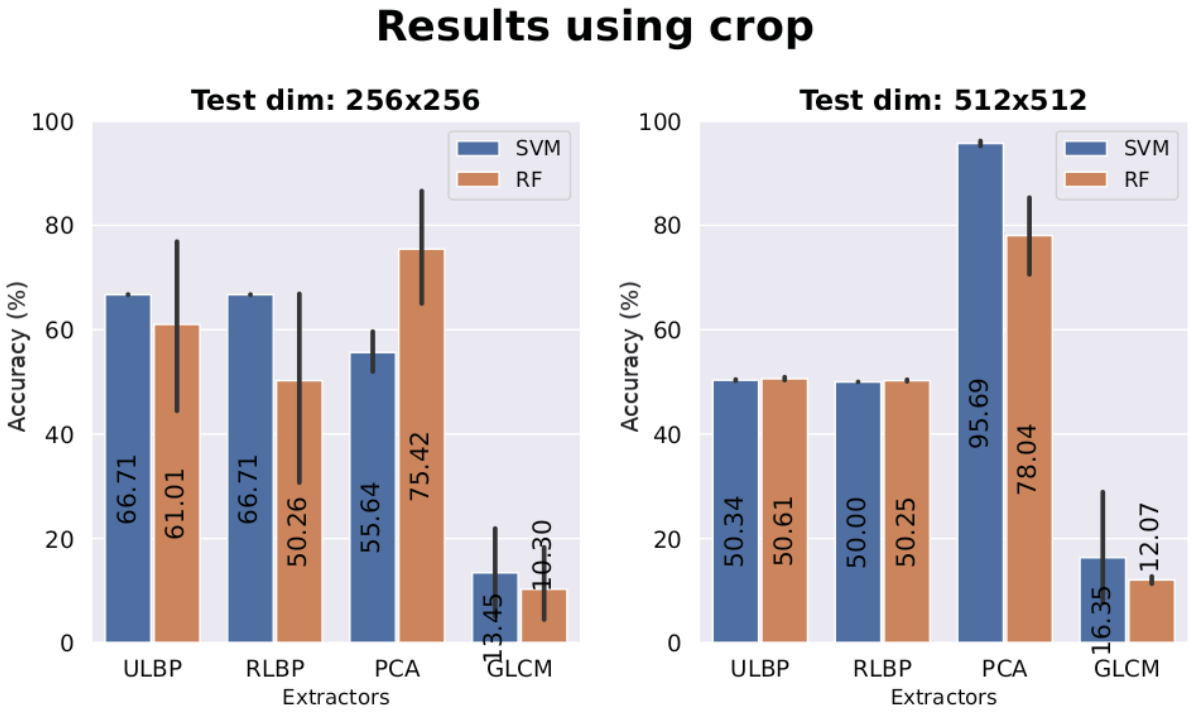


Figura 4.8: Classification results using 128x128 TMs for training and cropping the 128x128 middle values of test TMs. **1)** The left barchart shows the accuracy values classifying 256x256 TMs; and **2)** the right barchart shows the accuracy values classifying 512x512 TMs.

4.5 DILATION

Once the visual patterns present in the TMs are formed for few lines of pixels, important pixels to recognize patterns could be lost in the rescaling process. As described in Section 3.1,

dilation enhance the features of an image. So, as showed in Figure 4.9, we dilated the TMs before rescaling aiming to maintain the structure of pixels that forms the patterns after the preprocessing stage.

Depending on the communication pattern of the HPC application, the TM that represents it will be formed mostly by white pixels, while TMs from other HPC applications will contain more black pixels. To standardize the coloration, we chose to invert the colors of matrices formed mostly by white pixels. So that, regardless of the matrix class, the pattern it contains will be formed by white pixels. Next, we apply image dilation, followed by interpolation.

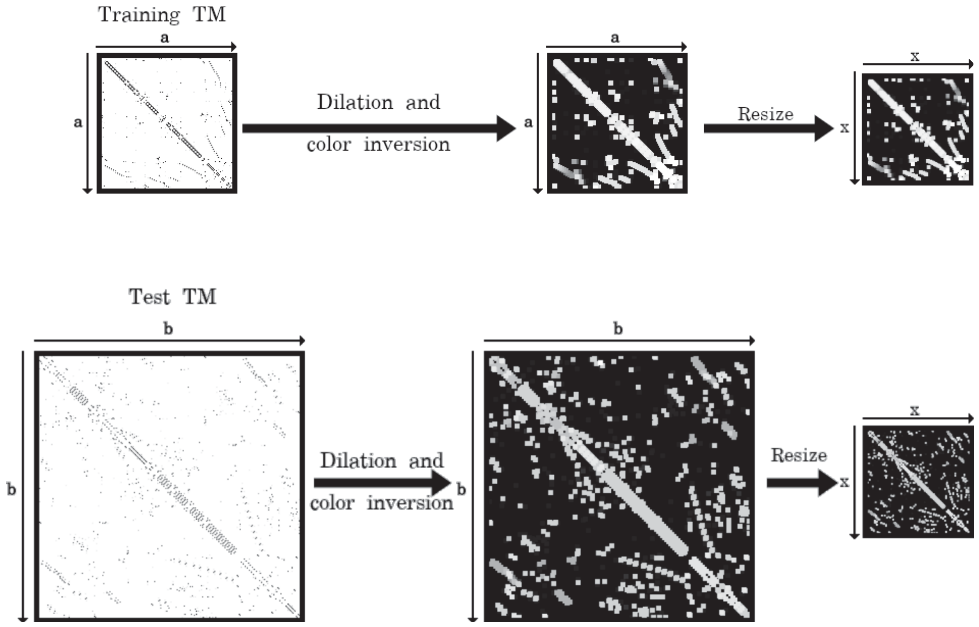


Figura 4.9: SCTReco's preprocessing step using dilation, inversion and resize.

Our first evaluation with dilation investigated four kernel sizes: 1x1, 3x3, 5x5, and 10x10; considered the classification accuracy value; and a real dataset containing three classes of TMs (bifu, multc, and cyl2d) with scales 128x128, used to train the SVM and RF classifiers, and 256x256, used to test. After the dilation process, the TMs was rescaled to 100x100, once this scale resulted in the best accuracy when we evaluated the interpolation algorithm. The results, exhibited in Table 4.3, contain the average results of ten executions and show that the kernel size 1x1 reached the highest accuracy value: 99.76%.

Kernel size	Classifier	ULBP (%)	RLBP (%)	PCA (%)	GLCM (%)
1x1	SVM	**99.76**	66.67	65.82	39.59
	RF	33.66	56.73	46.88	19.58
3x3	SVM	66.67	66.67	30.97	15.51
	RF	31.94	44.57	28.60	35.27
5x5	SVM	66.67	66.75	66.50	33.80
	RF	55.56	55.64	77.36	21.42
10x10	SVM	57.46	66.50	61.26	33.33
	RF	53.87	33.33	36.80	33.33

Tabela 4.3: Results of dilation method using kernel sizes 1x1, 3x3, 5x5, and 10x10.

Finally, we evaluated the dilation followed by rescaling to 100x100 using a real dataset containing two classes of TMs (bifu and cyl2d) with scales 128x128, used to train the SVM

and RF classifiers, and 512x512, used to test. We applied dilation with kernel size 1x1, and used ULBP, RLBP, PCA, and GLCM as feature extractors. The results, showed in Figure 4.10, containing the average results of ten executions, show that using SVM reached 85.23% of accuracy with ULBP as feature extractor; 50.21% with RLBP; 50.61% with PCA; and 53.31% with GLCM. Using RF, the accuracy value was 73.77% with ULBP; 66.50% with RLBP; 47.76% with PCA; and 22.20% with GLCM.

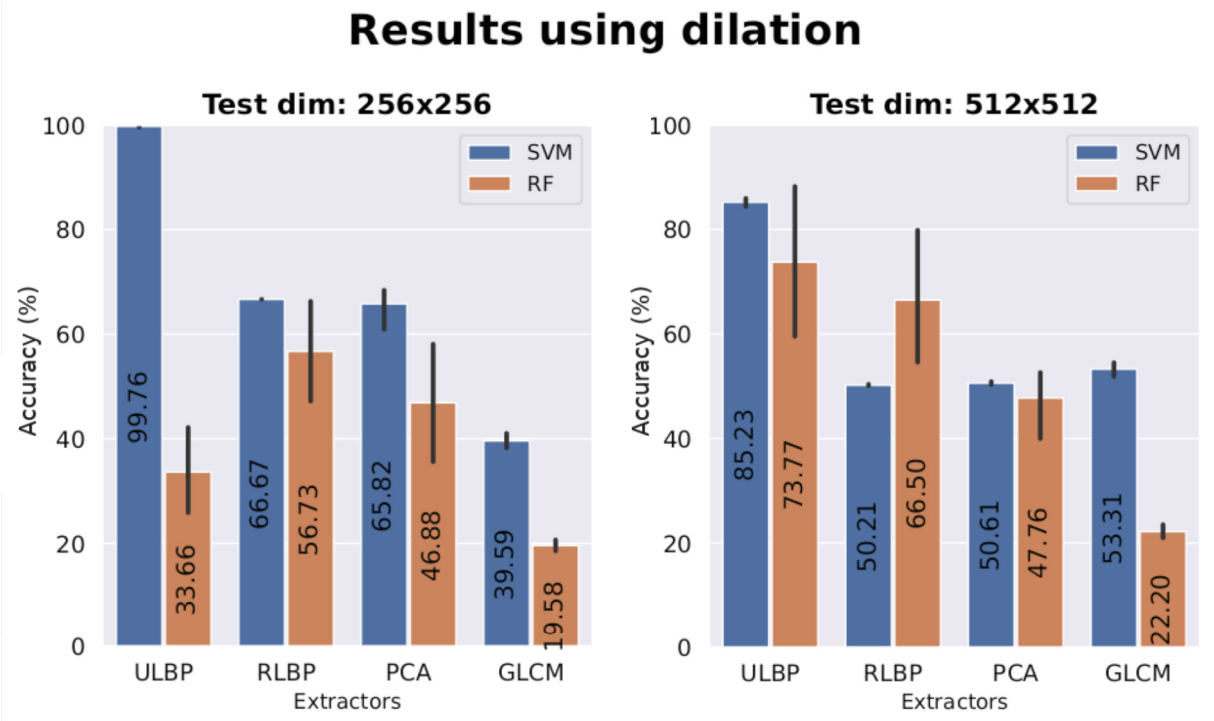


Figura 4.10: Classification results using 128x128 TMs for training, dilating the test TMs and rescaling them to 100x100. **1)** The left barchart shows the accuracy values classifying 256x256 TMs; and **2)** the right barchart shows the RF metric values classifying 512x512 TMs.

As we work with the visual textures expressed by the applications' TMs, we equipped SCTReco with the feature extractors GLCM and PCA. As described in Chapter 3, these two feature extractors works well with texture images. In aditional, we maintened the TReco's feature extractors: ULBP and RLBP as well as the TReco's classifiers: SVM and RF, once they work well with TMs. In the next section, we explain the evaluations made and show the results.

4.6 COMPARISON WITH TRECO

We evaluate TReco in the same environments used to evaluate SCTReco. The goal of this execution is compare the results between TReco and SCTReco and see the improvements. As mentioned in Section 3.4, TReco uses ULBP and RLBP as feature extractors; and RF and SVM as classifiers. Indeed, we compare the results with the feature extractors Eerman, Fahad, and Soysal.

The results, showed in Figure 4.11, containing the average results of ten executions, show that Fahad combined with RF reached 91.53% and 84.12% of accuracy classifying 256x256 TMs and 512x512 TMs, respectively. Using SVM to classify 256x256 TMs, ULBP, RLBP, Eerman and Soysal reached an accuracy value of 33.74%, 34.04%, 69.40%, 76.51%, and 66.96%, respectively. Using RF, the feature extractors ULBP, RLBP, Eerman, and Soysal reached 46.93%,

33.75%, 83.04%, 91.53%, and 65.36%, respectively. Classifying 512x512 TMs, SVM with ULBP, RLBP, Eerman, Fahad and Soysal reached 50.00%, 50.00%, 79.07%, 74.14%, and 77.00%, respectively. Using RF, the accuracy values was 50.06%, 50.06%, 87.62%, and 85.06% with ULBP, RLBP, Eerman and Soysal, respectively.

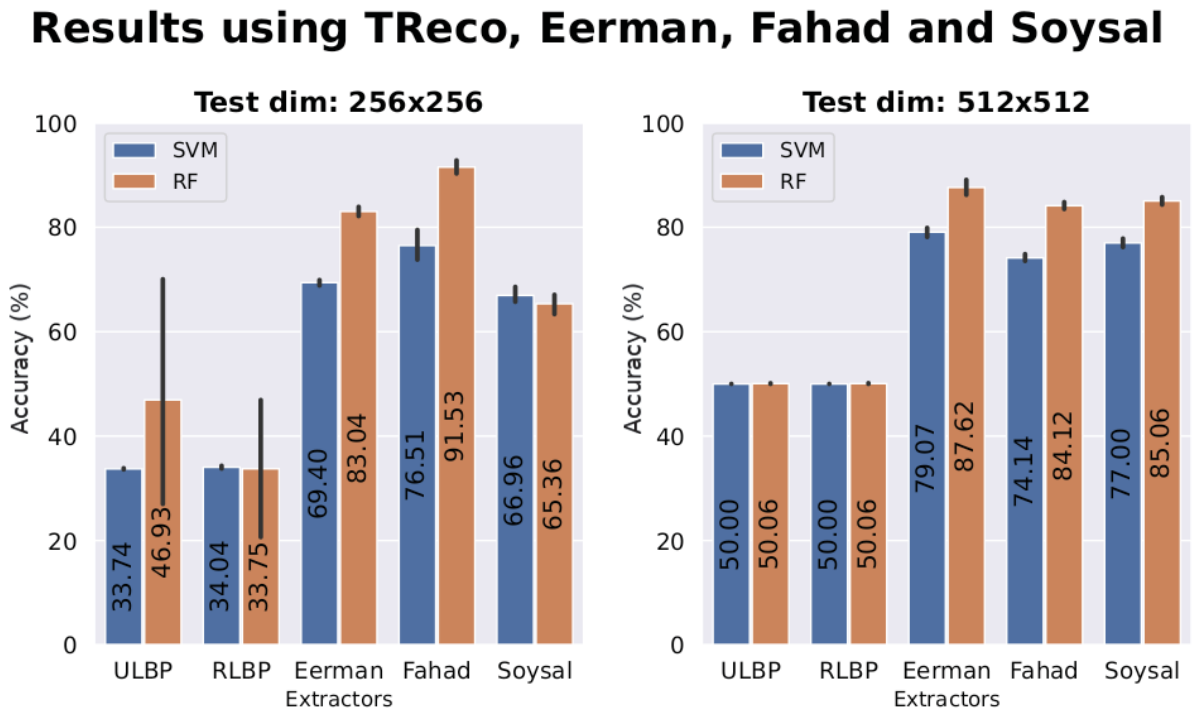


Figura 4.11: Classification results using 128x128 TMs for training, dilating the test TMs and rescaling them to 100x100. **1)** The left barchart shows the accuracy values classifying 256x256 TMs; and **2)** the right barchart shows the RF metric values classifying 512x512 TMs.

Analysing the results, we can see that SCTReco brings improvements over TReco. While TReco reached 46.93% of accuracy classifying 256x256 TMs, SCTReco reached 99.76%. In the same way, recognizing 512x512 TMs, TReco reached 50.06% of accuracy while SCTReco reached 85.23%. We can conclude that use dilation and interpolation as preprocessing algorithm increase the similarities between the TMs of different scales.

We can also observe that the use of the SVM and RF classifiers combined with the Eerman, Fahad and Soysal feature extractors achieved accuracy higher than that achieved by the TREco. So, we conclude that Eerman, Fahad and Soysal deal better with classifying TMs of different scales without the need for preprocessing algorithms when compared to ULBP and RLBP. But with the use of dilation and interpolation, ULBP combined with SVM achieved superior accuracy.

5 CONCLUSIONS

This work attempts to investigate how traffic matrix-based application classification methods behave in a scale-free environment. As traffic matrix-based applications have a pattern in their communications, ML techniques can be used to recognize them. We developed a framework named SCTRx, that combines the feature extractors RLBP, ULBP, GLCM, and PCA with SVM and RF. We used real TMs obtained from scientific applications to evaluate them, using the accuracy score as metrics. Indeed, SCTRx is equipped with a preprocessing method composed of dilation, interpolation, blur and crop.

We executed experiments to evaluate the combinations between preprocessing algorithms, feature extractors, and classifiers. The first experiment used 128x128 TMs to train the classifiers and TMs with 256x256 scale to test them. SCTRx reached 99.76% of accuracy using dilation and interpolation as preprocessing algorithm, ULBP as feature extractor, and SVM as the classifier. With the same TMs in the training step, SCTRx reached 85.23% of accuracy recognizing TMs with 512x512 scale, using the same preprocessing algorithm, feature extractor and classifier used in the previous evaluation.

SCTRx brings improvements over TReco, the previous framework for TMs recognition. TReco, using 128x128 TMs for training the classifiers, reached 46.93% of accuracy classifying 256x256 TMs, and 50.06% classifying 512x512 TMs. In comparison with the feature extractors Eerman, Fahad and Soysal, SCTRx got better results too, once the best accuracy result recognizing 256x256 TMs was reached by Fahad combined with RF classifier: 91.53%; and the best accuracy result recognizing 512x512 TMs was reached by Eerman combined with RF classifier: 87.62%.

So, we can conclude that ULBP and SVM are a prominent combination for recognizing TMs in a real-world scale-free environment, and preprocessing the TMs with dilation followed by interpolation increases the recognition capability. As future work, we plan to execute SCTRx using RGB TMs. Each type of network package can be represented as one RGB code. So, the color of each position of the TMs will be the result of all the packages exchanged among the nodes. The colors may be a feature that helps to distinguish the classes and increase the accuracy value with bigger TMs.

REFERÊNCIAS

Achour, S. e Nasri, W. (2012). A performance prediction approach for mpi routines on multi-clusters. Em *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, páginas 125–129. IEEE.

Akyildiz, I. F., Lee, A., Wang, P., Luo, M. e Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, 71:1–30.

Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N. e Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks. Em *NSDI*, volume 10, páginas 19–19.

Alickovic, E. e Subasi, A. (2018). Ensemble svm method for automatic sleep stage classification. *IEEE Transactions on Instrumentation and Measurement*, páginas 1258–1265.

Alsmadi, I., Khamaiseh, S. e Xu, D. (2016). Network parallelization in hpc clusters. Em *Computational Science and Computational Intelligence (CSCI), 2016 International Conference on*, páginas 584–589. IEEE.

Aouat, S., Ait-hammi, I. e Hamouchene, I. (2021). A new approach for texture segmentation based on the gray level co-occurrence matrix. *Multimedia Tools and Applications*.

Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W. et al. (2006). The landscape of parallel computing research: A view from berkeley. Relatório técnico, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

Badaghei, R., Hassanpour, H. e Askari, T. (2021). Detection of bikers without helmet using image texture and shape analysis. *International Journal of Engineering*.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S. et al. (1991). The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, páginas 63–73.

Benson, T., Akella, A. e Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. Em *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, páginas 267–280. ACM.

Bo, L., Zhenliu, Z. e Xiangfeng, W. (2012). A survey of hpc development. Em *2012 International Conference on Computer Science and Electronics Engineering*, páginas 103–106. IEEE.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. e Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, página 16.

Breiman, L. (2001). Random forests. *Machine learning*, páginas 5–32.

Cavalin, P. e Oliveira, L. (2017). A review of texture classification methods and databases. *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, páginas 1–8.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H. e Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. Em *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, páginas 44–54. IEEE.

Chen, J., Kellokumpu, V., Zhao, G. e Pietikainen, M. (2013). Rlbp: Robust local binary pattern.

Chowdhury, S. R., Bari, M. F., Ahmed, R. e Boutaba, R. (2014). Payless: A low cost network monitoring framework for software defined networks. Em *Network Operations and Management Symposium (NOMS), 2014 IEEE*, páginas 1–9. IEEE.

Colella, P. (2004). Defining software requirements for scientific computing.

Cortes, C. e Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, páginas 273–297.

Date, S., Abe, H., Khureltulga, D., Takahashi, K., Kido, Y., Watashiba, Y., Pongsakorn, U., Ichikawa, K., Yamanaka, H., Kawai, E. et al. (2015). An empirical study of sdn-accelerated hpc infrastructure for scientific research. Em *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, páginas 89–96. IEEE.

Dimond, R., Racaniere, S. e Pell, O. (2011). Accelerating large-scale hpc applications using fpgas. Em *Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on*, páginas 191–192. IEEE.

D'Orazio, V., Landis, S. T., Palmer, G. e Schrodt, P. (2014). Separating the wheat from the chaff: Applications of automated document classification using support vector machines. *Political Analysis*, página 224–242.

Eerman, J., Mahanti, A. e Arlitt, M. (2006). Internet traffic identification using machine learning techniques. Em *Proc. of the 49th IEEE Global Telecomm. Conf.(GLOBECOM)*, páginas 1–6. IEEE.

Erlacher, C., Jankowski, P., Blaschke, T., Paulus, G. e Anders, K.-H. (2017). A gpu-based parallelization approach to conduct spatially-explicit uncertainty and sensitivity analysis in the application domain of landscape assessment. *GI_Forum 2017,*, 1:44–58.

Erman, J., Arlitt, M. e Mahanti, A. (2006). Traffic classification using clustering algorithms. Em *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, páginas 281–286. ACM.

Erman, J., Mahanti, A., Arlitt, M., Cohen, I. e Williamson, C. (2007). Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, páginas 1194–1213.

Fahad, A., Tari, Z., Khalil, I., Almalawi, A. e Zomaya, A. (2014a). An optimal and stable feature selection approach for traffic classification based on multi-criterion fusion. *Future Generation Computer Systems*, página 156–169.

Fahad, A., Tari, Z., Khalil, I., Almalawi, A. e Zomaya, A. Y. (2014b). An optimal and stable feature selection approach for traffic classification based on multi-criterion fusion. *Future Generation Computer Systems*, páginas 156–169.

Faris, H., Hassonah, M., Al-Zoubi, A., Mirjalili, S. e Aljarah, I. (2017). A multi-verse optimizer approach for feature selection and optimizing svm parameters based on a robust system architecture. *Neural Computing and Applications*, páginas 1–15.

Farnaaz, N. e Jabbar, M. (2016). Random forest modeling for network intrusion detection system. *Procedia Computer Science*, páginas 213 – 217.

Feitelson, D. G. e Weil, A. M. (1998). Utilization and predictability in scheduling the ibm sp2 with backfilling. Em *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*, páginas 542–546. IEEE.

Feng, W.-c., Lin, H., Scogland, T. e Zhang, J. (2012). Opencl and the 13 dwarfs: a work in progress. Em *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, páginas 291–294. ACM.

F.R.S., K. P. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*.

Galante, G., De Bona, L. C. E., Mury, A. R., Schulze, B. e da Rosa Righi, R. (2016). An analysis of public clouds elasticity in the execution of scientific applications: a survey. *Journal of Grid Computing*, 14(2):193–216.

Gao, L., Chen, P. e Yu, S. (2016). Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Electron Device Letters*, páginas 870–873.

Gémieux, M., Savaria, Y., David, J.-P. e Zhu, G. (2017). A cache-coherent heterogeneous architecture for low latency real time applications. Em *Real-Time Distributed Computing (ISORC), 2017 IEEE 20th International Symposium on*, páginas 176–184. IEEE.

Giotis, K., Androulidakis, G. e Maglaris, V. (2014). Leveraging SDN for efficient anomaly detection and mitigation on legacy networks. Em *Proceedings of the third European Workshop on Software Defined Networks (EWSDN)*, páginas 85–90. IEEE.

Gong, Y., Wang, X., Malboubi, M., Wang, S., Xu, S. e Chuah, C.-N. (2015). Towards accurate online traffic matrix estimation in software-defined networks. Em *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, página 26. ACM.

Guo, Z. e Zhang, L. (2010). A completed modeling of local binary pattern operator for texture classification. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*.

Gupta, A. e Milojicic, D. (2011). Evaluation of hpc applications on cloud. Em *Open Cirrus Summit (OCS), 2011 Sixth*, páginas 22–26. IEEE.

Hamouchene, I., Aouat, S. e Lacheheb, H. (2014). Texture segmentation and matching using lbp operator and glcm matrix. *Studies in Computational Intelligence*.

He, L., Xu, C. e Luo, Y. (2016). vtc: Machine learning based traffic classification as a virtual network function. Em *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, páginas 53–56. ACM.

Heuveline, V. e Latt, J. (2007). The openlb project: an open source and object oriented implementation of lattice boltzmann methods. *International Journal of Modern Physics C*, 18(04):627–634.

Huang, W., Huang, Y., Wang, H., Liu, Y. e Shim, H. J. (2020). Local binary patterns and superpixel-based multiple kernels for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

Jain, N. (2016). *Optimization of communication intensive applications on HPC networks*. Tese de doutorado, University of Illinois at Urbana-Champaign.

Jamalian, S. e Rajaei, H. (2015). Data-intensive hpc tasks scheduling with sdn to enable hpc-as-a-service. Em *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, páginas 596–603. IEEE.

Jennings, B. e Stadler, R. (2015). Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, páginas 567–619.

Kalakech, M., Biela, P., Macaire, L. e Hamad, D. (2011). Constraint scores for semi-supervised feature selection: A comparative study. *Pattern Recognition Letters*, páginas 656 – 665.

Kamencay, P., Trnovszky, T., Benco, M., Hudec, R., Sykora, P. e Satnik, A. (2016). Accurate wild animal recognition using pca, lda and lbph. Em *2016 ELEKTRO*.

Kecman, V. e Wang, L. (2005). *Support Vector Machines – An Introduction*, páginas 1–47. Springer Berlin Heidelberg.

Koh, J., Lee, J. e Yoon, S. (2021). Single-image deblurring with neural networks: A comparative survey. *Computer Vision and Image Understanding*.

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S. e Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, páginas 14–76.

Krishna, S. V., Shrivastava, A. e Wagh, S. J. (2017). Sdn in high performance computing for scientific and business environment (sbe). Em *2017 International Conference on Computational Intelligence in Data Science(ICCIDS)*, páginas 1–8. IEEE.

Kulkarni, V. e Sinha, P. (2013). Random forest classifiers: A survey and future research directions. *International Journal of Advanced Computing*, páginas 1144–1153.

Kuster, L. (2017). *dCUDA: GPU Cluster Programming using IB Verbs*. Tese de doutorado, Department of Computer Science, ETH Zurich.

Lee, J., Tong, Z., Achalkar, K., Yuan, X. e Lang, M. (2016). Enhancing infiniband with openflow-style sdn capability. Em *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, páginas 36:1–36:12. IEEE Press.

Lee, J., Turner, Y., Lee, M., Popa, L., Banerjee, S., Kang, J.-M. e Sharma, P. (2014). Application-driven bandwidth guarantees in datacenters. Em *ACM SIGCOMM Computer Communication Review*, volume 44, páginas 467–478. ACM.

Lee, J. R., Canon, R. S., Declerck, T., Draney, B., Paul, D. e Skinner, D. (2018). Enhancing supercomputing with software defined networking. Em *2018 International Conference on Information Networking (ICOIN)*, páginas 571–576. IEEE.

Liu, Y., Li, Q., Du, B. e Farzaneh, M. (2021). Feature extraction and classification of surface discharges on an ice-covered insulator string during ac flashover using gray-level co-occurrence matrix. *Scientific Reports*.

Liu, Y., Li, W. e Li, Y. (2007). Network traffic classification using k-means clustering. Em *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, páginas 360–365. IEEE.

Ng, S. (2017). Principal component analysis to reduce dimension on digital image. *Procedia Computer Science*.

Ning, Q., Zhu, J. e Chen, C. (2018). Very fast semantic image segmentation using hierarchical dilation and feature refining. *Cognitive Computation*.

Nugteren, C. (2017). Clblast: A tuned opencl blas library. *arXiv preprint arXiv:1705.05249*.

Nunes, B. A. A., Mendonca, M., Nguyen, X., Obraczka, K. e Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys Tutorials*, páginas 1617–1634.

Ojala, T., Pietikainen, M. e Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, páginas 971–987.

Ojala, T., Pietikäinen, M. e Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions.

ONF, O. (2013). Sdn architecture overview. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf`. Accessed 12 mar. 2017.

Osin, A. V. e Sheluhin, O. I. (2019). Network quality operation prediction based on machine learning algorithms. Em *2019 Systems of Signals Generating and Processing in the Field of on Board Communications*, páginas 1–4.

Parsania, P. e Virparia, D. (2016). A comparative analysis of image interpolation algorithms. *IJARCCE*.

Perez, L. e Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *CoRR*.

Polezhaev, P., Shukhman, A. e Ushakov, Y. (2014). Network resource control system for hpc based on sdn. Em *International Conference on Next Generation Wired/Wireless Networking*, páginas 219–230. Springer.

Prabhakar, R., Zhang, Y., Koeplinger, D., Feldman, M., Zhao, T., Hadjis, S., Pedram, A., Kozyrakis, C. e Olukotun, K. (2017). Plasticine: A reconfigurable architecture for parallel paterns. Em *Proceedings of the 44th Annual International Symposium on Computer Architecture*, páginas 389–402. ACM.

Pradhan, A. (2012). Support vector machine-a survey. *IJETAE*, 2.

Rasras, R. J., Zahran, B., Sara, M. R. e AlQadi, Z. (2021). Developing digital signal clustering method using local binary pattern histogram. *International Journal of Electrical and Computer Engineering*.

Renner, T., Thamsen, L. e Kao, O. (2015). Network-aware resource management for scalable data analytics frameworks. Em *Big Data (Big Data), 2015 IEEE International Conference on*, páginas 2793–2800. IEEE.

Righi, R., Rodrigues, V., Costa, C., Galante, G., de Bona, L. C. E. e Ferreto, T. (2016). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Transactions on Cloud Computing*, 4(1):6–19.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, páginas 1–39.

Roughan, M., Sen, S., Spatscheck, O. e Duffield, N. (2004). Class-of-service mapping for qos: A statistical signature-based approach to ip traffic classification. Em *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, páginas 135–148. ACM.

Rubin, E., Levy, E., Barak, A. e Ben-Nun, T. (2014). Maps: Optimizing massively parallel applications using device-level memory abstraction. páginas 1–22.

Sakr, S., Liu, A., Batista, D. M. e Alomari, M. (2011). A survey of large scale data management approaches in cloud environments. *Communications Surveys & Tutorials, IEEE*, 13(3):311–336.

Samuel, A., Zahavi, E. e Keslassy, I. (2017). Routing keys. Em *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, páginas 9–16. IEEE Computer Society.

Sarwinda, D., Paradisa, R. H., Bustamam, A. e Anggia, P. (2021). Deep learning in image classification using residual network (resnet) variants for detection of colorectal cancer. *Procedia Computer Science*.

Scornet, E. (2017). Tuning parameters in random forests. *ESAIM: Procs*, páginas 144–162.

Sheikhpour, R., Sarram, M. A., Gharaghani, S. e Chahooki, M. A. Z. (2017). A survey on semi-supervised feature selection methods. *Pattern Recogn.*, páginas 141–158.

Simmons, J. M. (2014). *Optical network design and planning*. Springer.

Soysal, M. e Schmidt, E. (2010a). Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Perform. Eval.*, páginas 451–467.

Soysal, M. e Schmidt, E. G. (2010b). Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*, páginas 451–467.

Spatscheck, O., Levchenko, K., Kreibich, C. e Savage, S. (2014). Machine learning based traffic classification using low level features and statistical analysis.

Springer, P. (2011). Berkeley's dwarfs on cuda. *RWTH Aachen University, Tech. Rep*.

Srivastava, G., Singh, M., Kumar, P. e Singh, J. (2016). Internet traffic classification: A survey. Em *Recent Advances in Mathematics, Statistics and Computer Science*, páginas 611–620. World Scientific.

Srivastava, M., Siddiqui, J. e Ali, M. (2020). Image copy detection based on local binary pattern and svm classifier. *Cybernetics and Information Technologies*.

Takahashi, K., Khureltulga, D., Munkhdorj, B., Kido, Y., Date, S., Yamanaka, H., Kawai, E. e Shimojo, S. (2015). Concept and design of sdn-enhanced mpi framework. Em *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, páginas 109–110. IEEE.

Tangherloni, A., Nobile, M. S., Besozzi, D., Mauri, G. e Cazzaniga, P. (2017). Lassie: simulating large-scale models of biochemical systems on gpus. *BMC bioinformatics*, 18(1):246.

Tibrewala, R., Pedoia, V., Bucknor, M. e Majumdar, S. (2020). Principal component analysis of simultaneous pet-mri reveals patterns of bone–cartilage interactions in osteoarthritis. *Journal of Magnetic Resonance Imaging*.

Trestian, R., Muntean, G. M. e Katrinis, K. (2013). Micetrap: Scalable traffic engineering of datacenter mice flows using openflow. Em *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, páginas 904–907.

Trois, C. (2017). *Communication patterns abstractions for programming SDN to optimize high-performance computing applications*. Tese de doutorado, Federal University of Paraná, Curitiba, Brazil.

Trois, C., Bona, L. C., Oliveira, L. S., Martinello, M., Harewood-Gill, D., Del Fabro, M. D., Nejabati, R., Simeonidou, D., Lima, J. C. e Stein, B. (2018). Exploring textures in traffic matrices to classify data center communications. Em *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, páginas 1123–1130. IEEE.

Trois, C., de Bona, L. C. E., Fabro, M. D. D., Martinello, M., Bidkar, S., Nejabati, R. e Simeonidou, D. (2017). Softening up the network for scientific applications. Em *Parallel, Distributed and Network-Based Processing (PDP), 25th Euromicro Inte. Conference on*, páginas 410–418. IEEE.

Trois, C., de Bona, L. C. E., Fabro, M. D. D. D. e Martinello, M. (2016). Carving Software-Defined Networks for Scientific Applications with SpateN. Em *41st Conference on Local Computer Networks (LCN)*, páginas 606–610. IEEE.

Van Adrichem, N. L., Doerr, C., Kuipers, F. et al. (2014). Opennetmon: Network monitoring in openflow software-defined networks. Em *Network Operations and Management Symposium (NOMS), 2014 IEEE*, páginas 1–8. IEEE.

Vasiljevic, I., Chakrabarti, A. e Shakhnarovich, G. (2017). Examining the impact of blur on recognition by convolutional networks.

Vassiliev, A. V. (2017). Scalable opencl fpga computing evolution. Em *Proceedings of the 5th International Workshop on OpenCL*, página 22. ACM.

Vetter, J. S. e Mueller, F. (2002). Communication characteristics of large-scale scientific applications for contemporary cluster architectures. Em *Proceedings 16th International Parallel and Distributed Processing Symposium*, páginas 10 pp–. IEEE.

Wang, J., Rubin, N., Sidelnik, A. e Yalamanchili, S. (2016). Laperm: Locality aware scheduler for dynamic parallelism on gpus. Em *Proceedings of the 43rd International Symposium on Computer Architecture*, páginas 583–595. IEEE Press.

Wang, M., Cui, Y., Wang, X., Xiao, S. e Jiang, J. (2018). Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, páginas 92–99.

Wang, N., Ho, K., Pavlou, G. e Howarth, M. (2008). An overview of routing optimization for internet traffic engineering. *Communications Surveys & Tutorials, IEEE*, 10(1):36–56.

Wang, P. e Chung, A. C. S. (2018). Focal dice loss and image dilation for brain tumor segmentation. Springer International Publishing.

Werner, S., Navaridas, J. e Luján, M. (2017). Designing low-power, low-latency networks-on-chip by optimally combining electrical and optical links. Em *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, páginas 265–276. IEEE.

Wu, Z., Lu, K., Wang, X. e Chi, W. (2016). Alleviating network congestion for hpc clusters with fat-tree interconnection leveraging software-defined networking. Em *Systems and Informatics (ICSAI), 2016 3rd International Conference on*, páginas 808–813. IEEE.

Xu, J., Wu, Q., Zhang, J. e Tang, Z. (2012). Object detection based on co-occurrence gmulbp features. IEEE Computer Society.

Yang, C. (2018). Anomaly network traffic detection algorithm based on information entropy measurement under the cloud computing environment. *Cluster Computing*.

Yao, H., Li, H., Liu, C., Xiong, M., Zeng, D. e Li, G. (2016). Joint optimization of vm placement and rule placement towards energy efficient software-defined data centers. Em *Computer and Information Technology (CIT), 2016 IEEE International Conference on*, páginas 204–209. IEEE.

Ye, J., Cheng, X., Zhu, J., Feng, L. e Song, L. (2018). A ddos attack detection method based on svm in software defined network. *Security and Communication Networks*, páginas 1–8.

Yu, J., Liu, G., Dong, W. e Li, X. (2017). Using locality-enhanced distributed memory cache to accelerate applications on high performance computers. Em *Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2017 IEEE 3rd International Conference on*, páginas 160–166. IEEE.

Yu, R., Yang, Y., Yang, L., Han, G. e Move, O. A. (2016). Raq–a random forest approach for predicting air quality in urban sensing systems. Em *Sensors*.

Yu, Y., Qian, C. e Li, X. (2014). Distributed and collaborative traffic monitoring in software defined networks. Em *Proceedings of the third workshop on Hot topics in software defined networking*, páginas 85–90. ACM.

Zhang, D., Tang, D., Tang, L., Dai, R., Chen, J. e Zhu, N. (2019). Pca-svm-based approach of detecting low-rate dos attack. Em *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, páginas 1163–1170.

Zhang, H., Lu, G., Qassrawi, M. T., Zhang, Y. e Yu, X. (2012). Feature selection for optimizing traffic classification. *Computer Communications*, páginas 1457–1471.

Zhang, S., Wu, Y. e Chang, J. (2020). Survey of image recognition algorithms. Em *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*.

Zhang, Y. e Ge, Z. (2005). Finding critical traffic matrices. Em *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, páginas 188–197. IEEE.

Zhao, Y., Jia, W., Hu, R.-X. e Min, H. (2013a). Completed robust local binary pattern for texture classification. *Neurocomputing*, páginas 68–76.

Zhao, Y., Jia, W., Hu, R.-X. e Min, H. (2013b). Completed robust local binary pattern for texture classification. *Neurocomputing*.